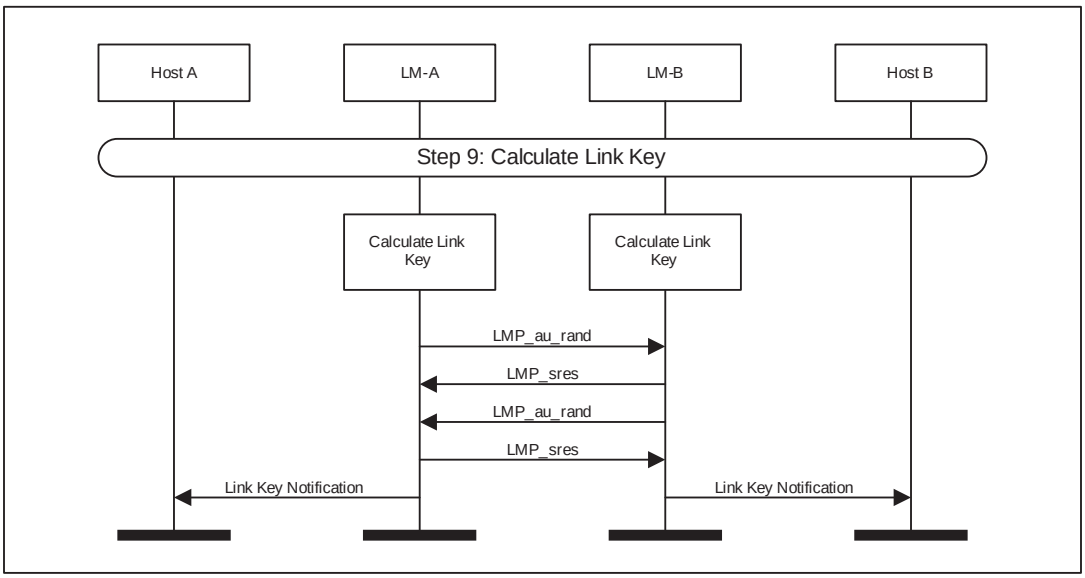



*Figure 4.19: Calculate Link Key*

### 4.2.20 Enable Encryption

Once the link key has been notified to the host, the Authentication Requested command will complete with an Authentication Complete event. The host can then turn on encryption using the standard methods.

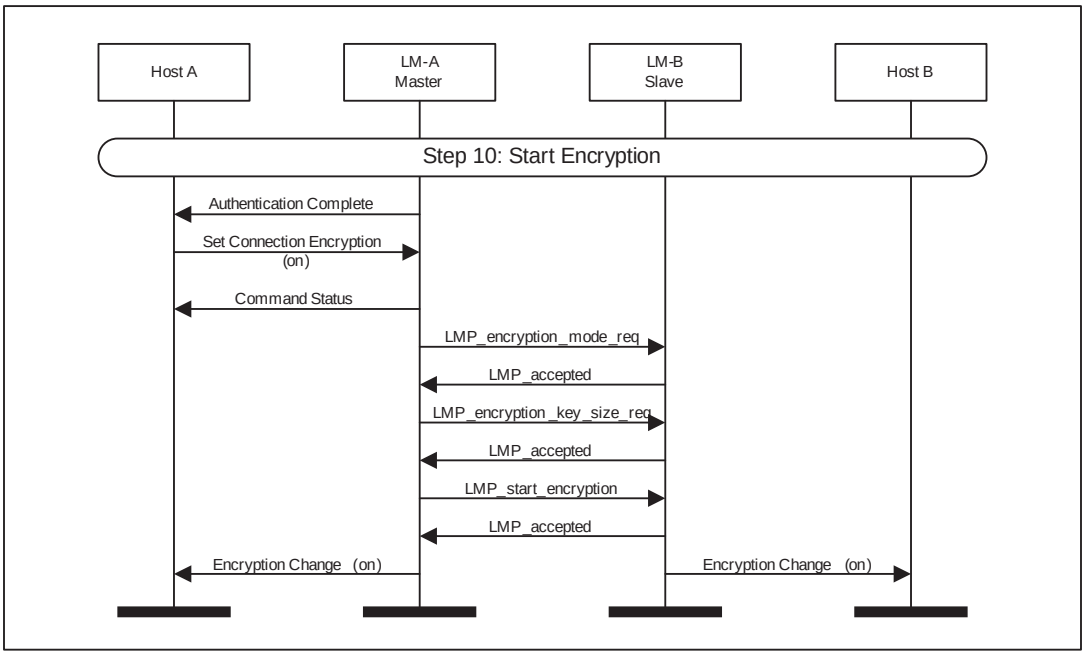



*Figure 4.20: Start Encryption*

### 4.2.21 L2CAP Connection Response

If this simple pairing was triggered by an L2CAP Connection Request, then only after all of the above steps have completed can the L2CAP Connection Response message be sent.

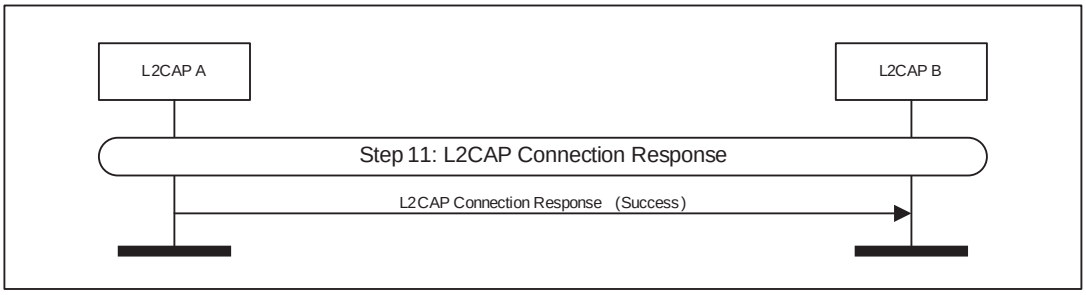



*Figure 4.21: L2CAP Connection Response*

## 4.3 LINK SUPERVISION TIMEOUT CHANGED EVENT

When enabled by the Host, the slave generates an HCI_Link_Supervision_Timeout_Changed event after the LMP_supervision_timeout PDU is received.

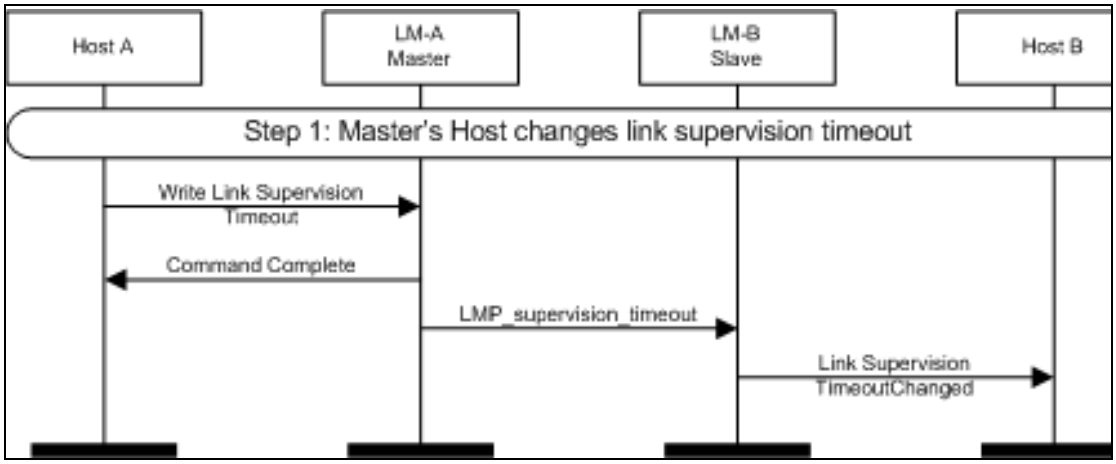



*Figure 4.22: Link Supervision Timeout Event*

## 4.4 SET CONNECTION ENCRYPTION

**Step 1:** The host may at any time turn on encryption using the HCI_Set_Connection_Encryption command. This command can be originated from either the master or slave sides. Only the master side is shown in Figure 4.23 on page 908. If this command is sent from a slave, the only difference is that the LMP_encryption_mode_req PDU will be sent from the slave. The LMP_encryption_key_size_req and LMP_start_encryption_req PDUs will always be requested from the master. (See Figure 4.23 on page 908.)

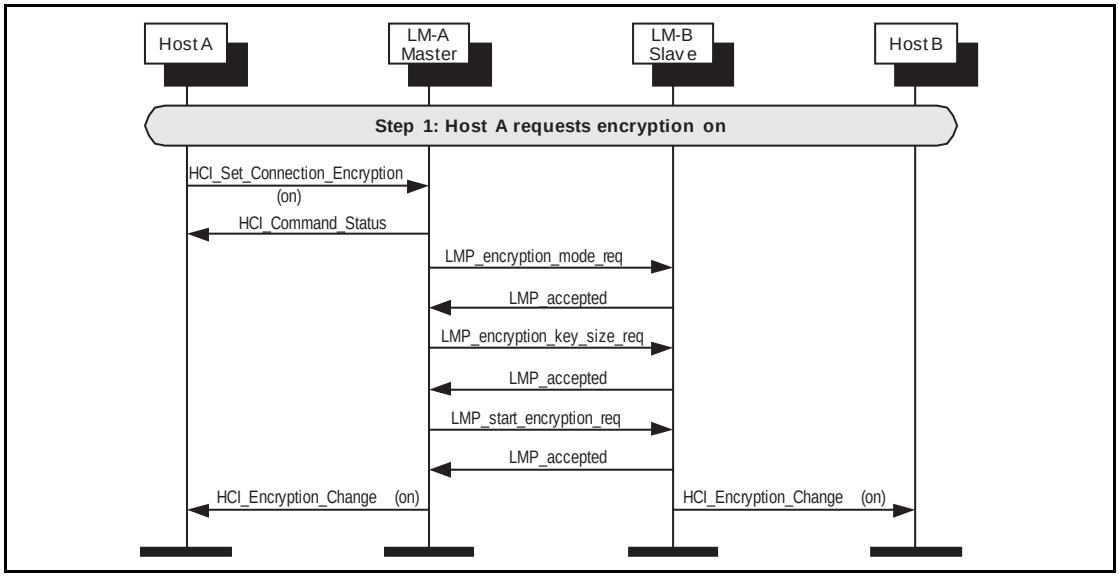



*Figure 4.23: Encryption requested.*

**Step 2:** To terminate the use of encryption, The HCI_Set_Connection_Encryption command is used. (See Figure 4.24 on page 908.)

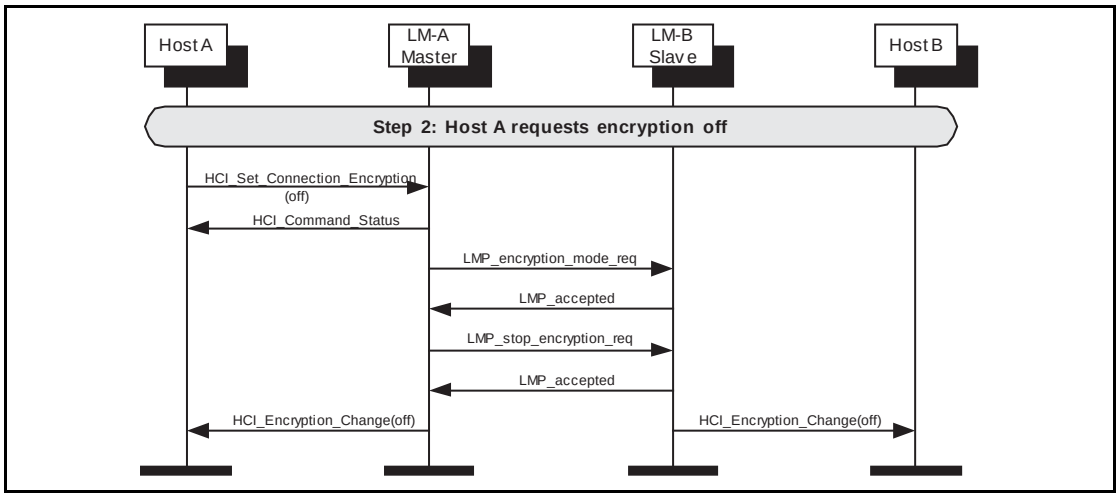



*Figure 4.24: Encryption off requested.*

## 4.5 CHANGE CONNECTION LINK KEY

**Step 1:** The master host (Host A) may change the connection link key using the HCI_Change_Connection_Link_Key command. A new link key will be generated and the hosts will be notified of this new link key. (See Figure 4.25 on page 909.)

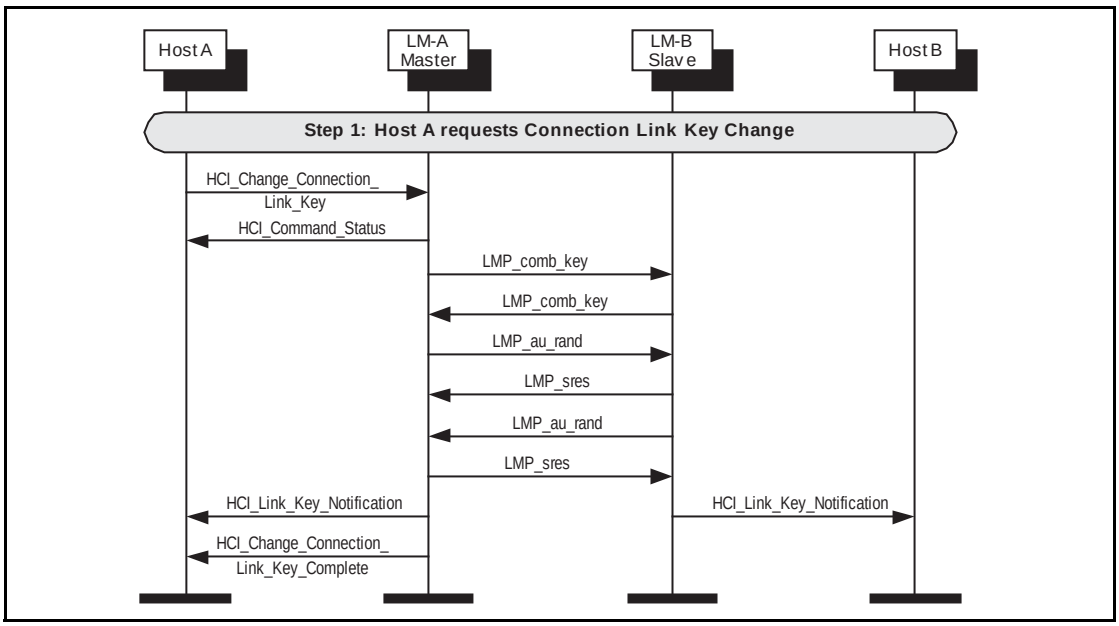



*Figure 4.25: Change connection link key.*

## 4.6 CHANGE CONNECTION LINK KEY WITH ENCRYPTION PAUSE AND RESUME

Step 1: The master host (Host A) may change the connection link key using the HCI_Change_Connection_Link_Key command. A new link key will be generated and the hosts will be notified of this new link key. Encryption will then be paused and resumed, immediately using this new link key to generate a new encryption key. (See Figure 4.26 on page 910.)

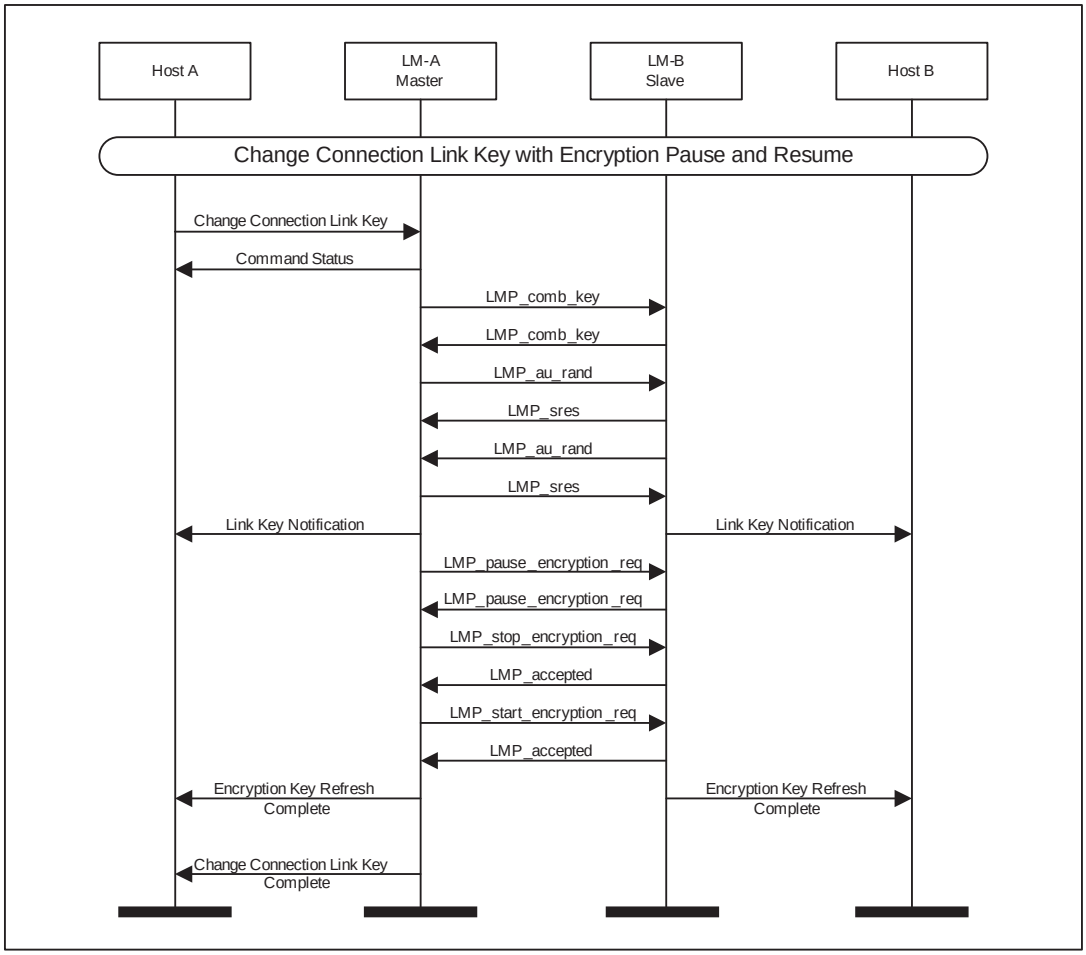



*Figure 4.26: Change Connection Link Key with Encryption Pause Resume*

## 4.7 MASTER LINK KEY

**Step 1:** The host changes to a Master Link Key from a Semi-permanent Link Key using the HCI_Master_Link_Key command. (See Figure 4.27 on page 911.)

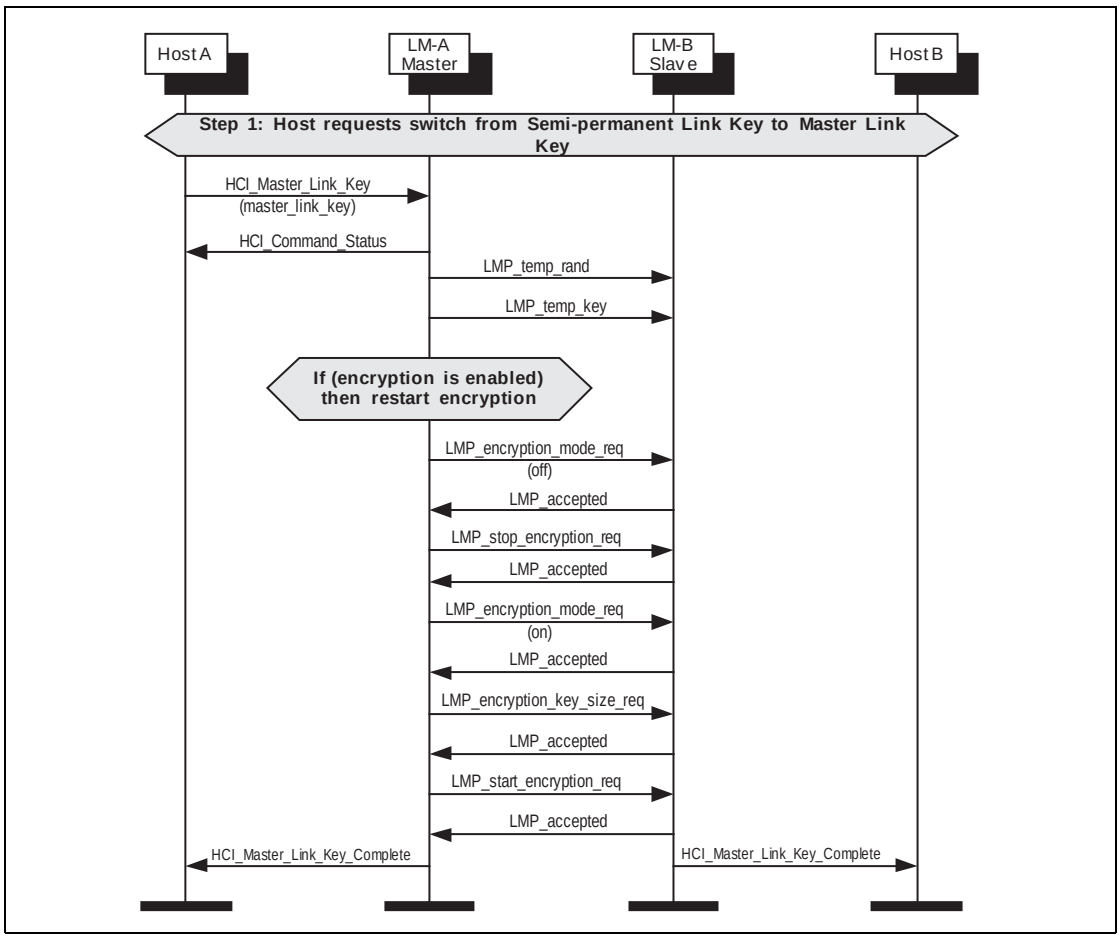



*Figure 4.27: Change to master link key.*

**Step 2:** The host changes to a Semi-permanent Link Key from a Master Link Key using the HCI_Master_Link_Key command. (See Figure 4.28 on page 912.)

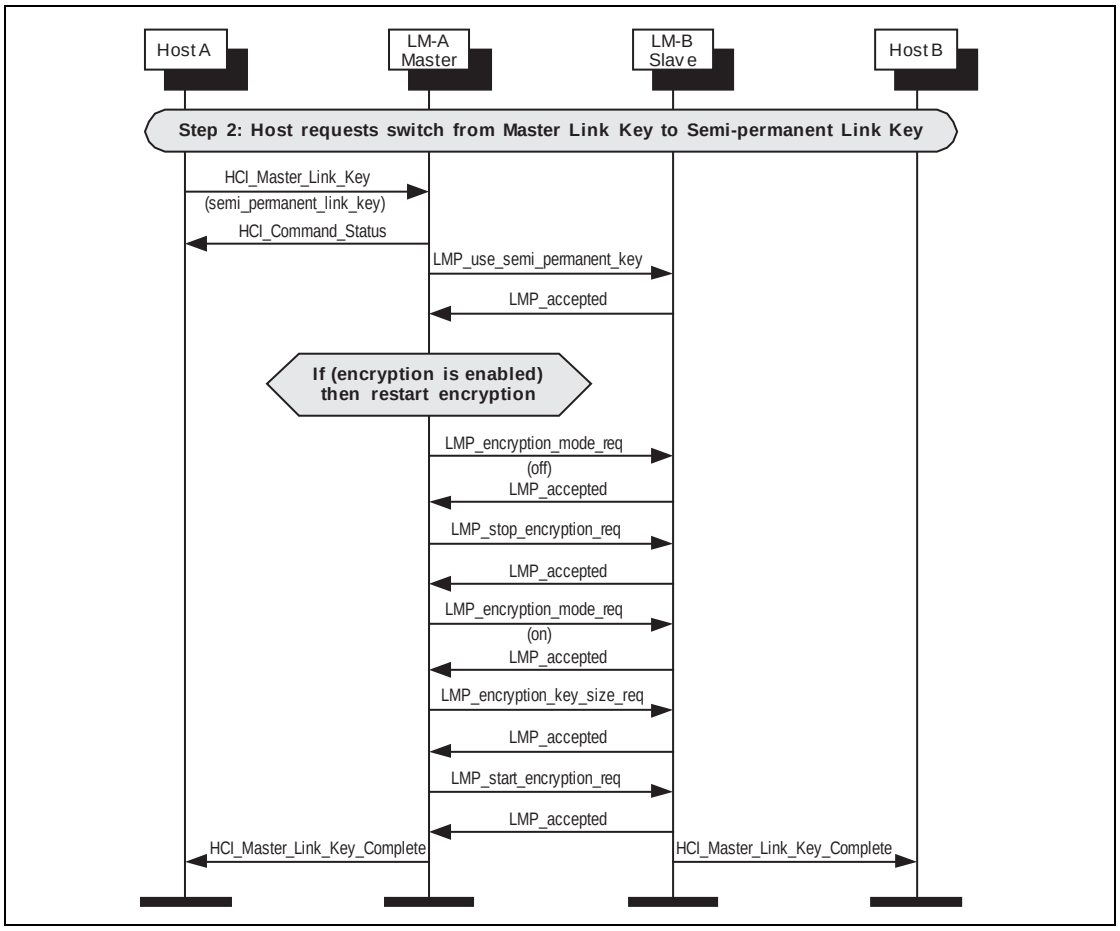



*Figure 4.28: Change to semi permanent link key.*

## 4.8 READ REMOTE SUPPORTED FEATURES

Using the HCI_Read_Remote_Supported_Features command the supported LMP Features of a remote device can be read. (See Figure 4.29 on page 913.)

If the remote supported features have been obtained previously then the Controller may return them without sending any LMP PDUs.

**Step 1:** The host requests the supported features of a remote device.

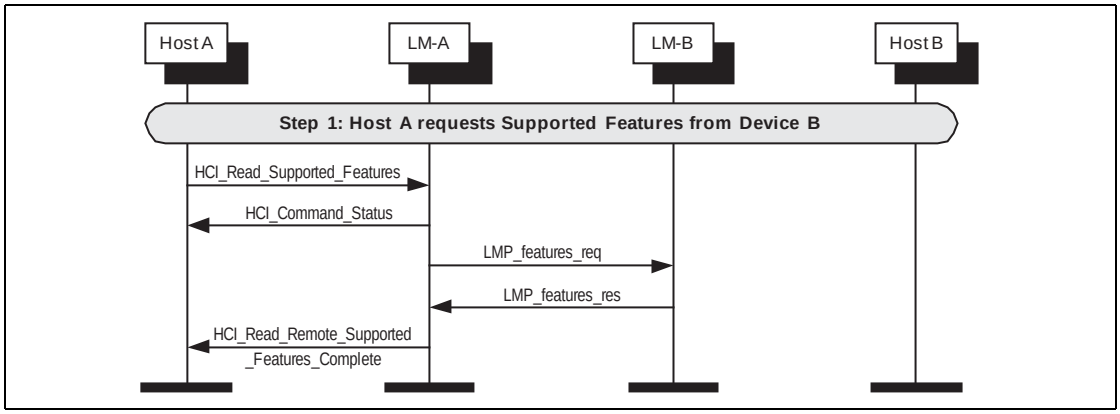



*Figure 4.29: Read remote supported features.*

## 4.9 READ REMOTE EXTENDED FEATURES

Using the HCI_Read_Remote_Extended_Features command the extended LMP features of a remote device can be read. (See Figure 4.30 on page 913.)

If the remote extended features have been obtained previously then the Controller may return them without sending any LMP PDUs.

**Step 1:** The host requests the extended features of a remote device.

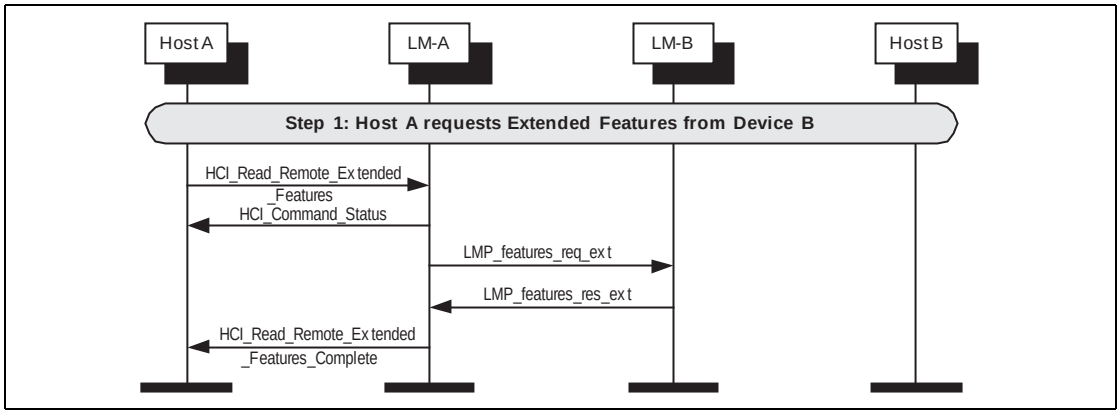



*Figure 4.30: Read remote extended features.*

## 4.10 READ CLOCK OFFSET

Using the HCI_Read_Clock_Offset command the device acting as the master can read the Clock Offset of a slave. The Clock Offset can be used to speed up the paging procedure in a later connection attempt. If the command is requested from the slave device, the Controller will directly return a Command Status event and a Read Clock Offset Complete event without sending any LMP PDUs. (See Figure 4.31 on page 914.)

**Step 1:** The host requests the clock offset of a remote device.

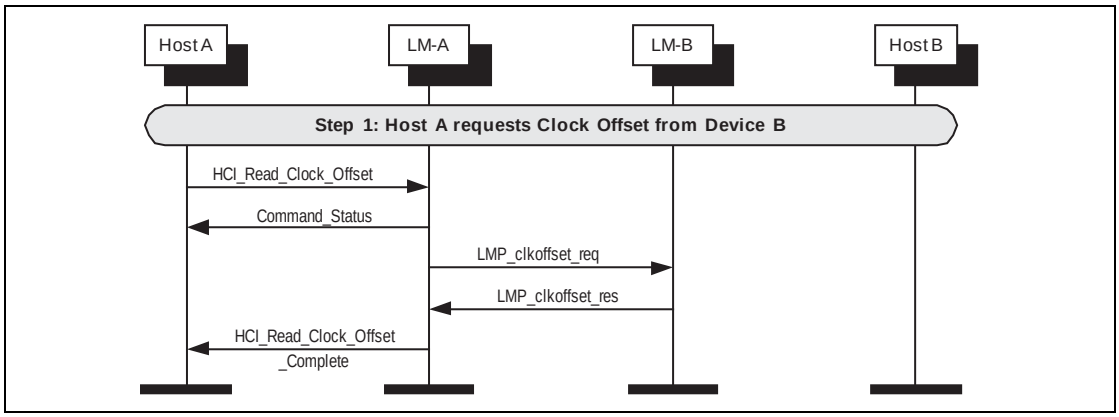



*Figure 4.31: Read clock offset.*

## 4.11 ROLE SWITCH ON AN ENCRYPTED LINK USING ENCRYPTION PAUSE AND RESUME

The HCI_Switch_Role command can be used to explicitly switch the current master / slave role of the local device with the specified device. The master host (A) requests a role switch with a slave. This will first pause encryption, and then send the switch request, and the slave will respond with the slot offset and accepted. The role switch is performed by doing the TDD switch and piconet switch. Encryption is resumed, and finally an HCI_Role_Change event is sent on both sides. (Section 4.12 on page 915)

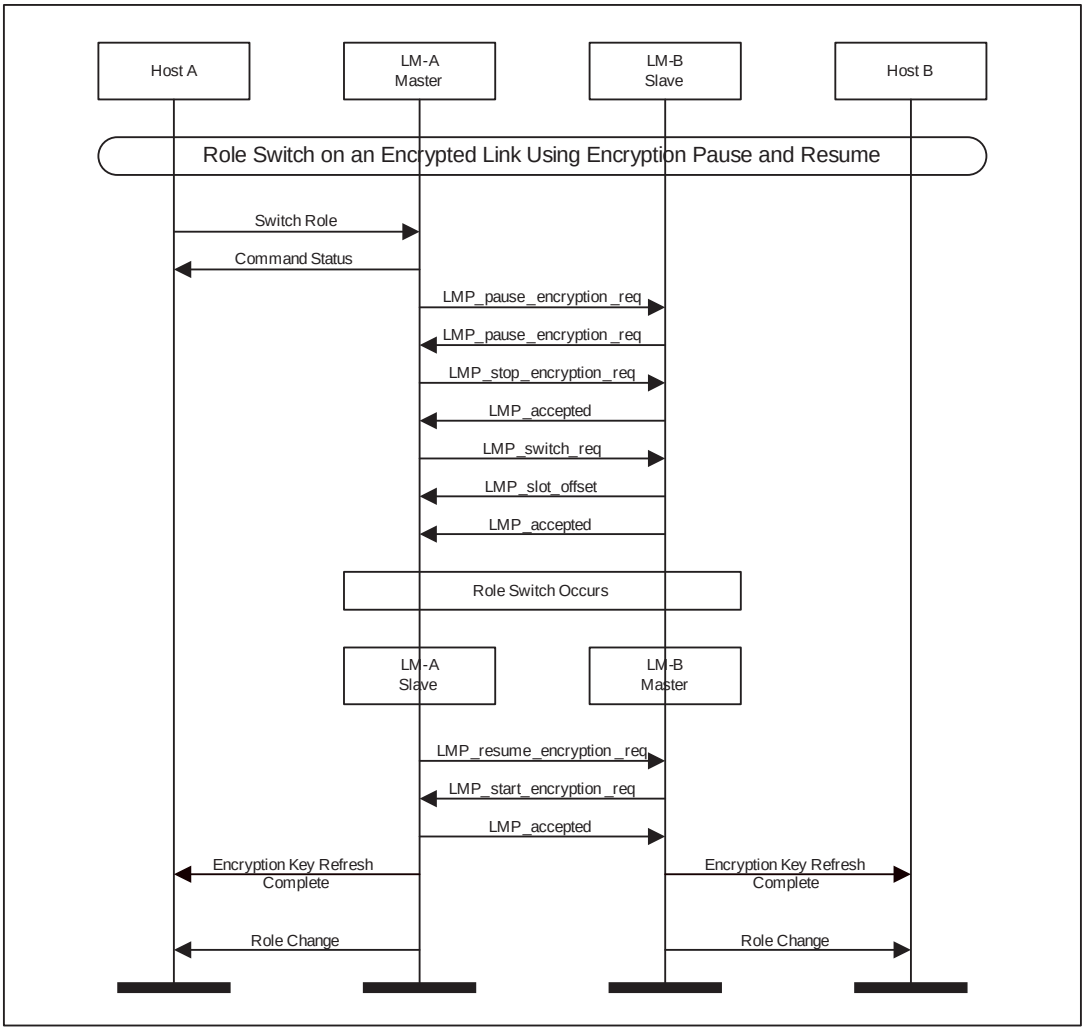



*Figure 4.32: Role Switch on an Encrypted link using encryption pause and resume.*

## 4.12 REFRESHING ENCRYPTION KEYS

The HCI_Refresh_Encryption_Key command may be used by the master's Host to explicitly pause and resuming encryption to refresh the encryption key. After encryption is resumed an HCI_Encryption_Key_Freshness event is sent on both sides. (Section 4.13 on page 902).

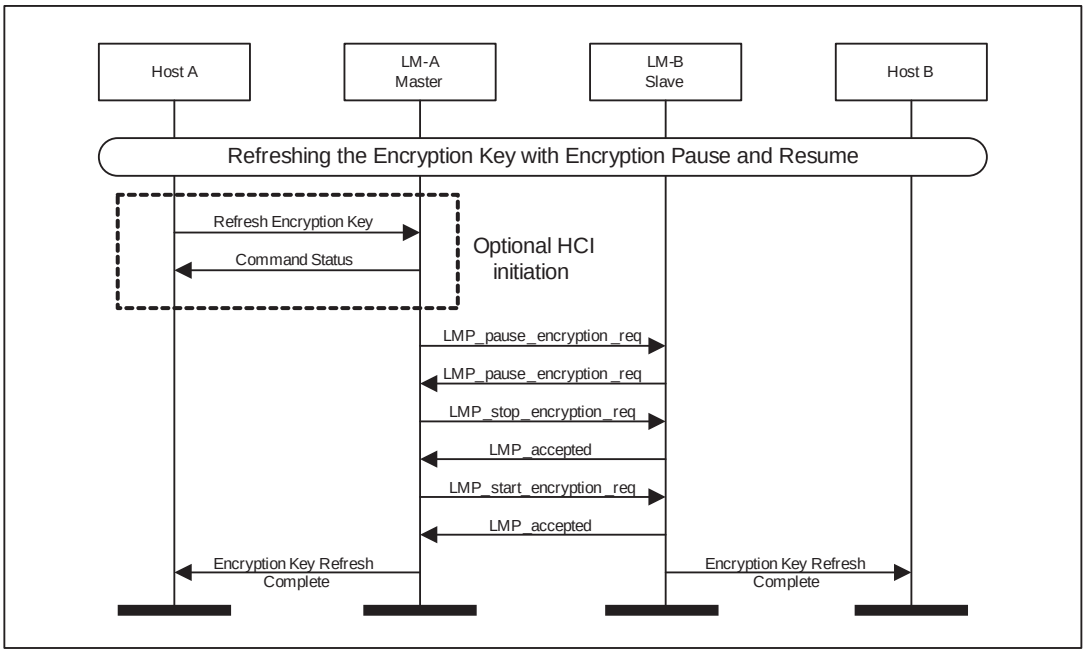



*Figure 4.33: Refreshing encryption keys using encryption pause and resume*

## 4.13 READ REMOTE VERSION INFORMATION

Using the HCI_Read_Remote_Version_Information command the version information of a remote device can be read. (See Figure 4.34 on page 916.)

If the remote version information has been obtained previously then the Controller may return them without sending any LMP PDUs.

**Step 1:** The host requests the version information of a remote device.

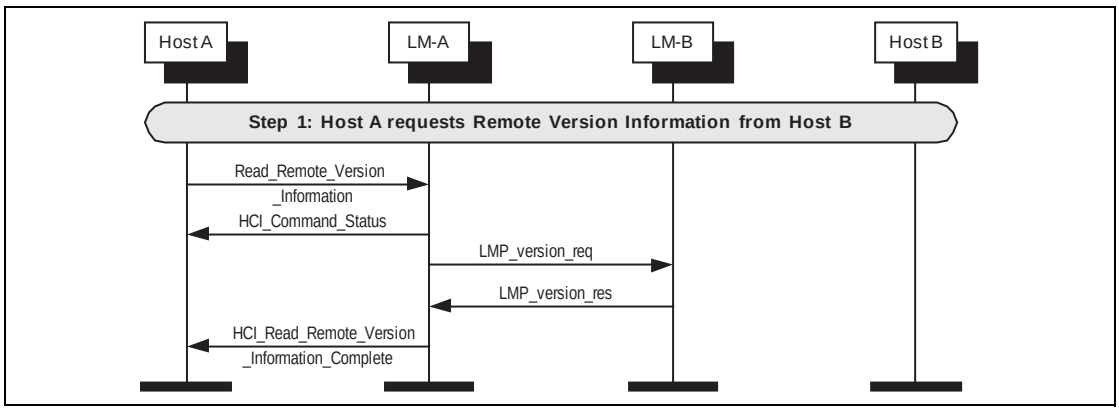



*Figure 4.34: Read remote version information.*

## 4.14 QOS SETUP

Using the HCI_Flow_Specification command the Quality of Service (QoS) and Flow Specification requirements of a connection can be notified to a Controller.

The Controller may then change the quality of service parameters with a remote device. (See Figure 4.35 on page 917.)

**Step 1:** The host sends QoS parameters to a remote device.

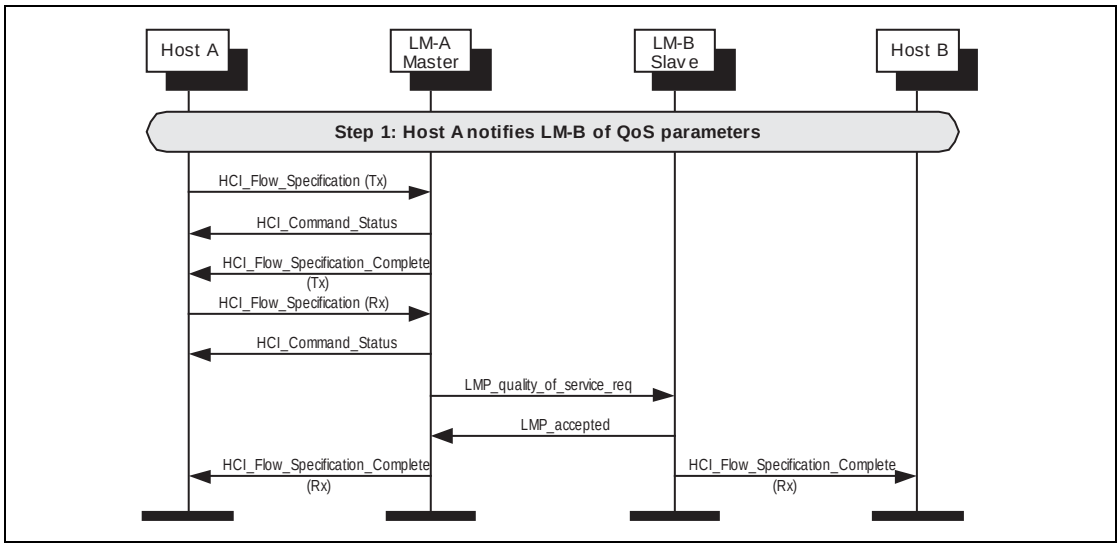



*Figure 4.35: QoS flow specification.*

## 4.15 SWITCH ROLE

The HCI_Switch_Role command can be used to explicitly switch the current master / slave role of the local device with the specified device.

**Step 1a:** The master host (A) requests a role switch with a slave. This will send the switch request, and the slave will respond with the slot offset and accepted. (See Figure 4.36 on page 917.)

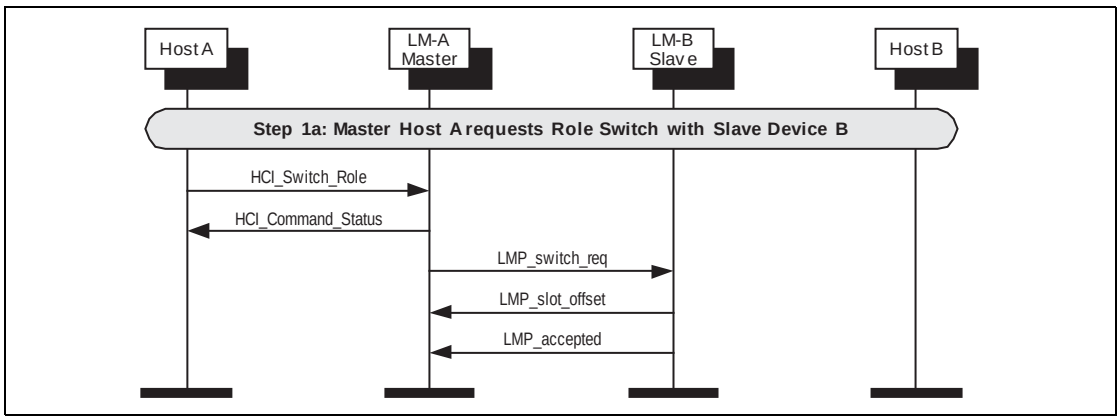



*Figure 4.36: Master requests role switch.*

**Step 1b:** The slave host (B) requests a role switch with a master. This will send the slot offset and the switch request, and the master will respond with a LMP_accepted PDU. (See Figure 4.37 on page 918.)

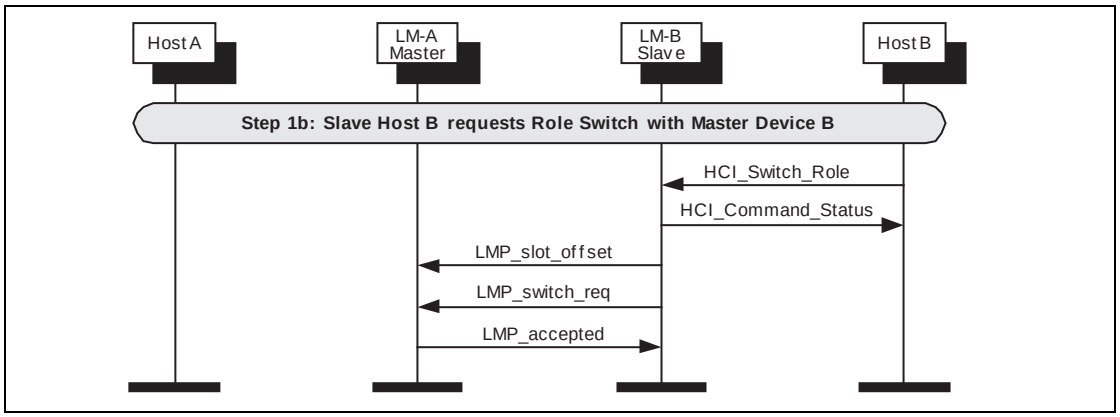



*Figure 4.37: Slave requests role switch.*

**Step 2:** The role switch is performed by doing the TDD switch and piconet switch. Finally an HCI_Role_Change event is sent on both sides.
(See Figure 4.38 on page 918.)

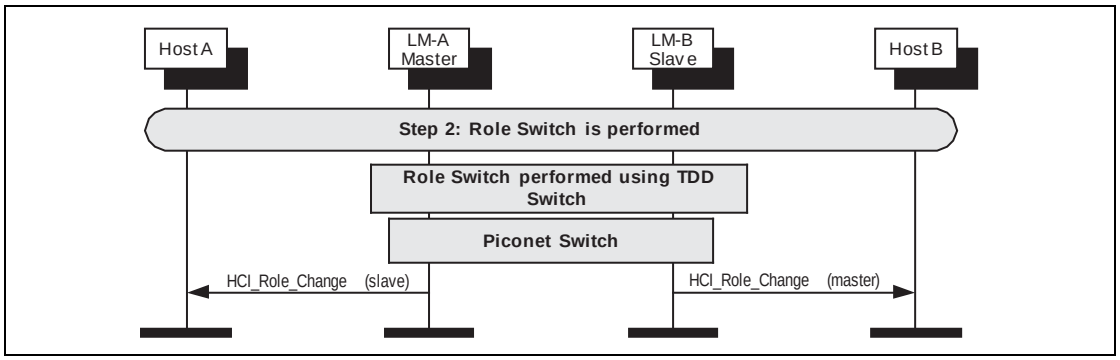



*Figure 4.38: Role switch is performed.*

## 4.16 AMP PHYSICAL LINK CREATION AND DISCONNECT

A flow diagram of the AMP link establishment and detachment of a connection between two devices is shown in Figure 4.39.

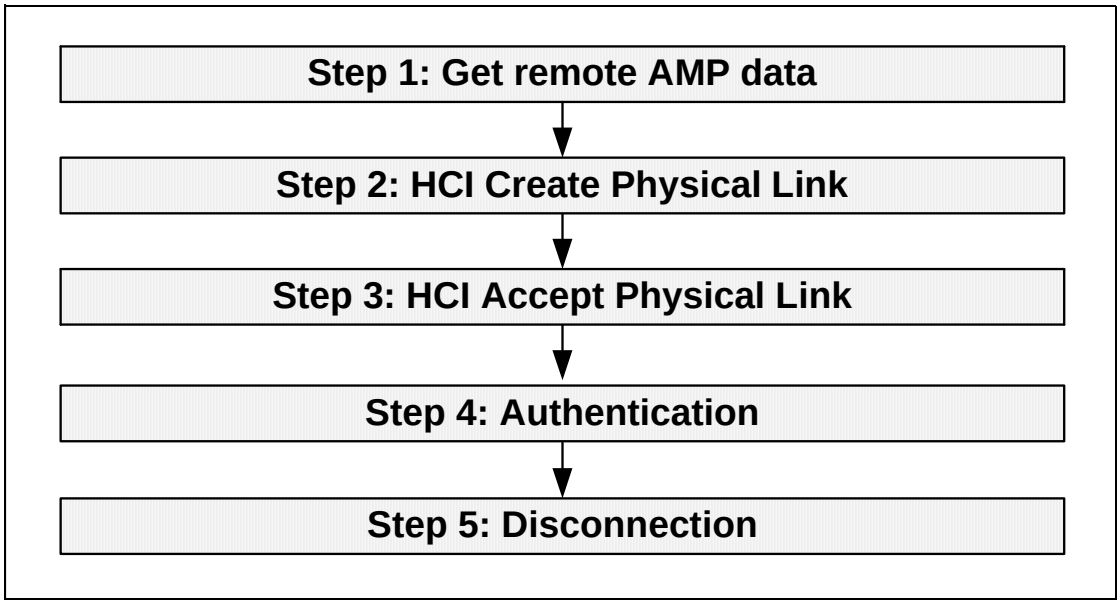



*Figure 4.39: Overview diagram for Physical Link creation and disconnection*

### 4.16.1 Physical Link Establishment

The process of establishing a Physical Link consists of 4 steps.

**Step 1:** Host A uses the BR/EDR Controller to request AMP_Info data from Host B in order to create an AMP connection. Host B gets this information from PAL B using HCI Read Local AMP Info. Host B returns the information to Host A over the BR/EDR radio.

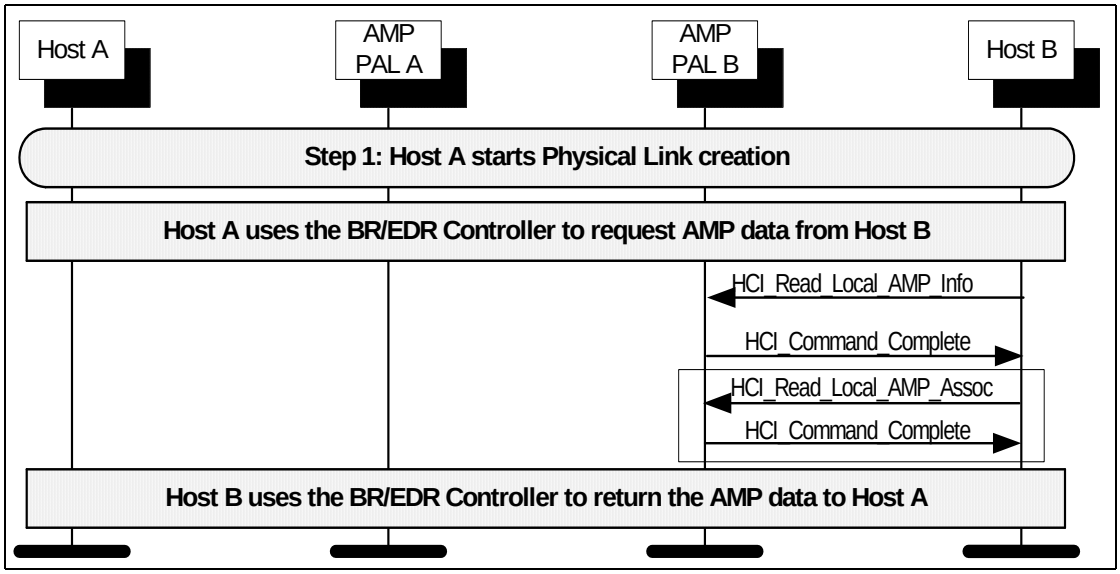



*Figure 4.40: Get remote AMP data*

**Step 2:** Host A sends an HCI Create Physical Link command to its AMP Controller. AMP Controller A determines which channel will be used, and provides channel information to its Host. AMP Controller A joins or creates that channel.

Note use of dotted rectangles for zero or multiple uses of read or write AMP_Assoc.

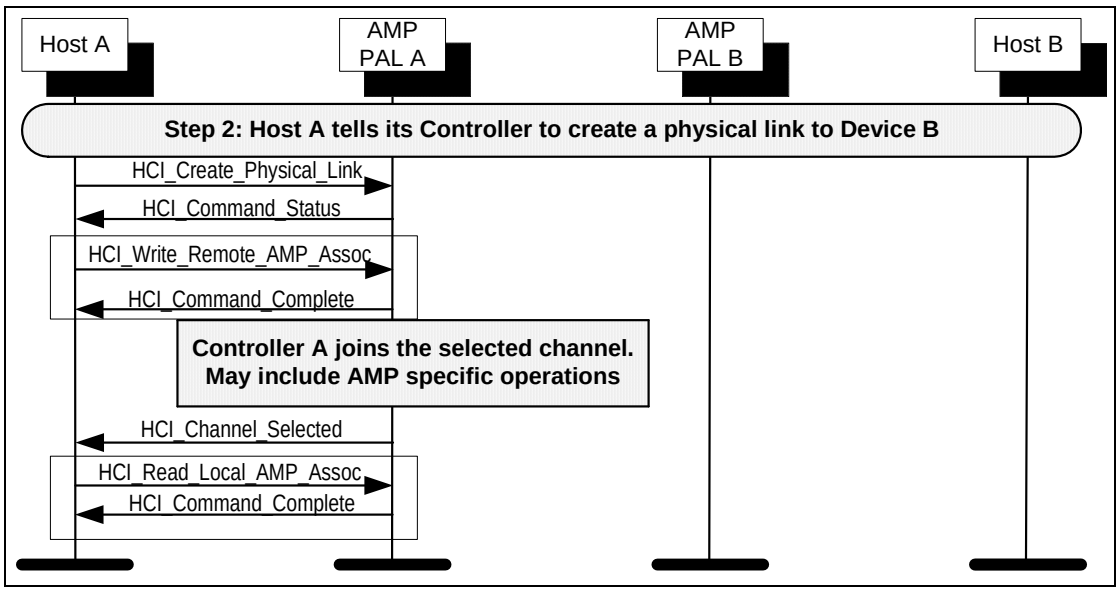



*Figure 4.41: Host A tells its Controller to create the physical link*

**Step 3:** Host A uses the BR/EDR radio to request a physical link to B. Host B tells its AMP Controller to accept a physical link from Device A.

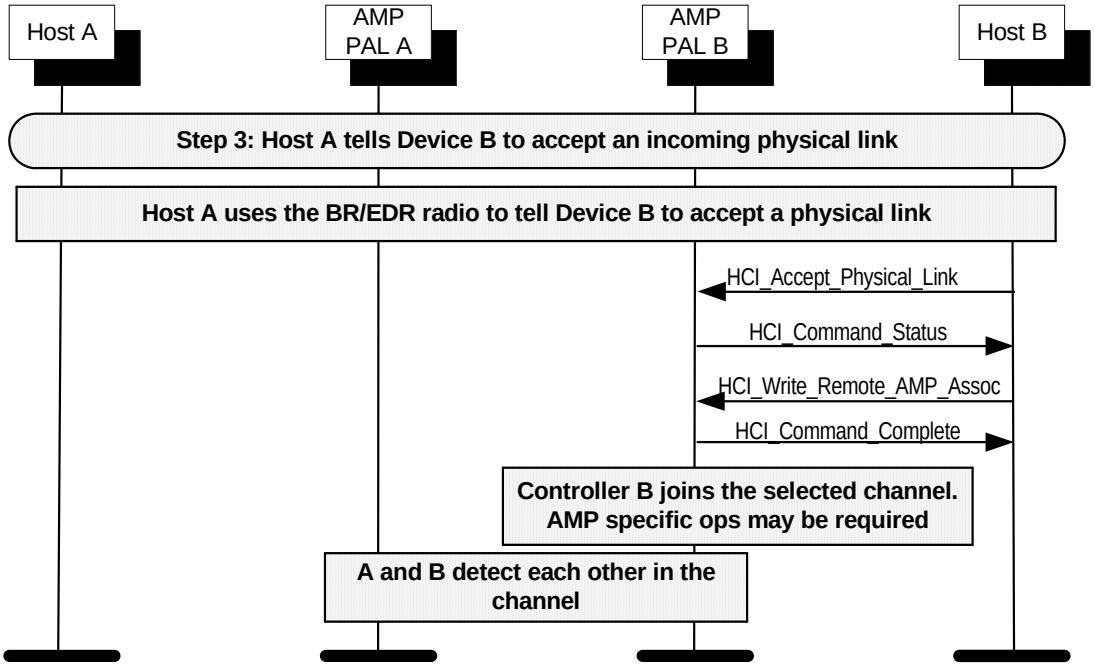



*Figure 4.42: Host A tells Host B to accept an incoming physical link*

**Step 4:** The AMP Controllers perform security operations. When the hand-shake is complete, both devices inform their Hosts that the link is ready for use.

Note that the mechanism used to transport the authentication messages is defined by each PAL.

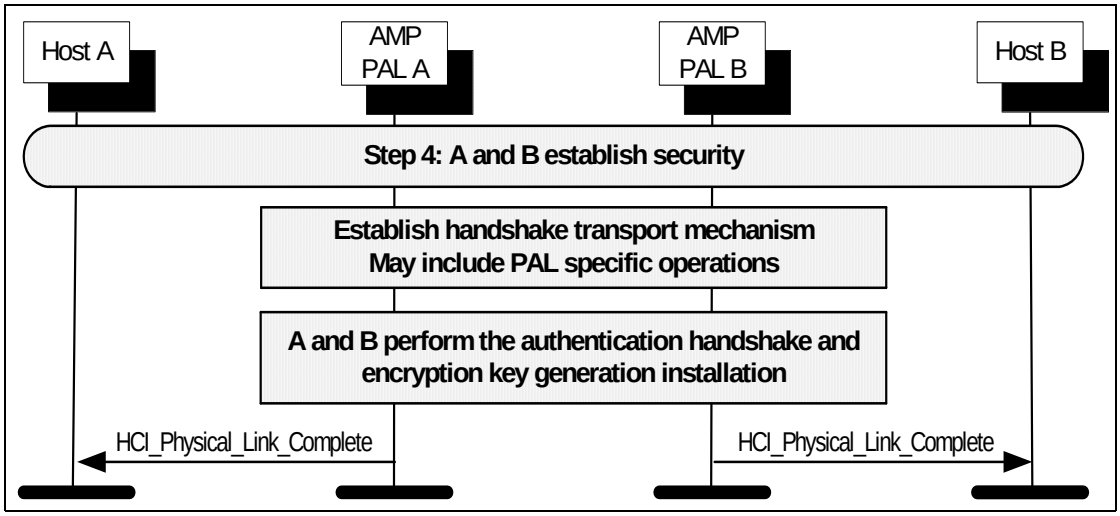



*Figure 4.43: A and B establish AMP physical link security*

**Step 5:** Host A sends a Physical Link Disconnect command to its AMP Controller. AMP Controllers A and B do whatever AMP specific action is required to meet the request. Each AMP Controller sends the Physical Link Disconnect Complete event to its host. Both devices can now leave the AMP channel. Note that disconnection of any active Logical Links is not shown in Figure 4.44.

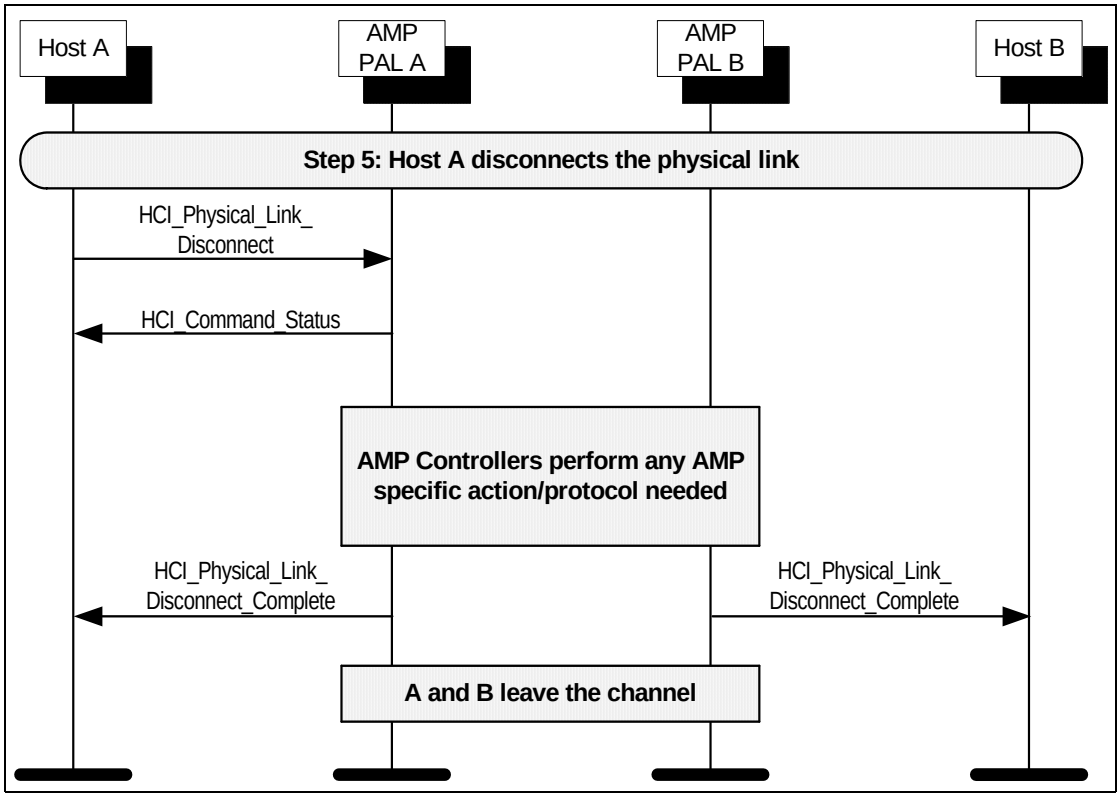



*Figure 4.44: Host A disconnects the physical link*

**Step 6:** During normal operation the AMP Controller may alert its Host to issues on the link.

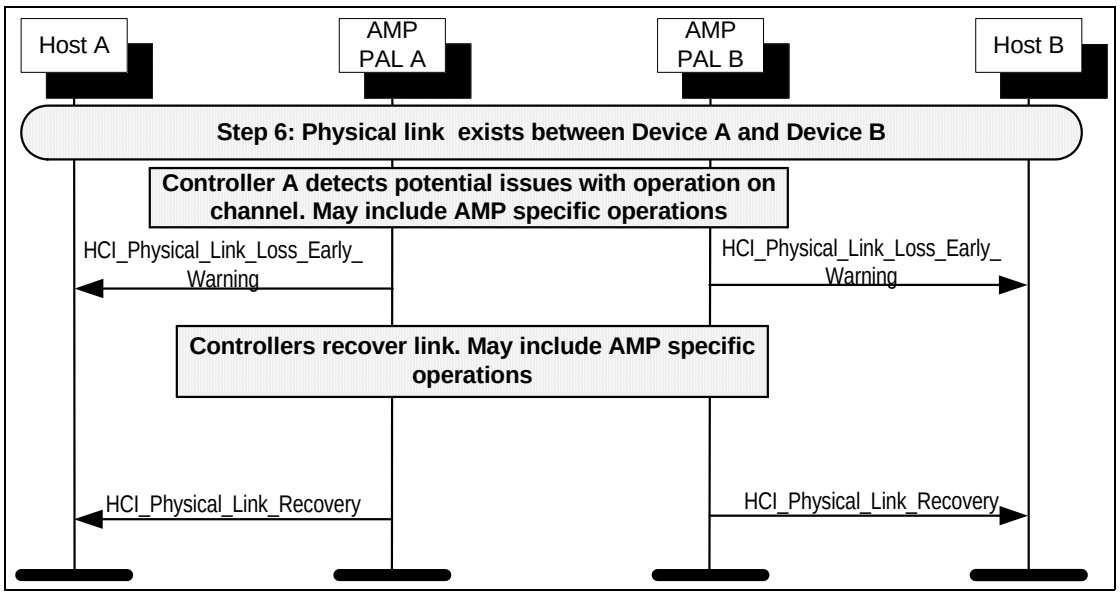



*Figure 4.45: Controller reports to its Host of Link Loss*

### 4.16.2 Logical Link Creation

**Step 1:** Host A sends a Logical Link Create command to its AMP Controller. AMP Controllers A and B do whatever AMP specific action is required to meet the bandwidth request. Each AMP Controller sends the Logical Link Creation Complete event to its host. Both devices can now pass data traffic over the AMP channel.

A flow diagram of the establishment of a Guaranteed logical connection between two devices is shown in Figure 4.46.

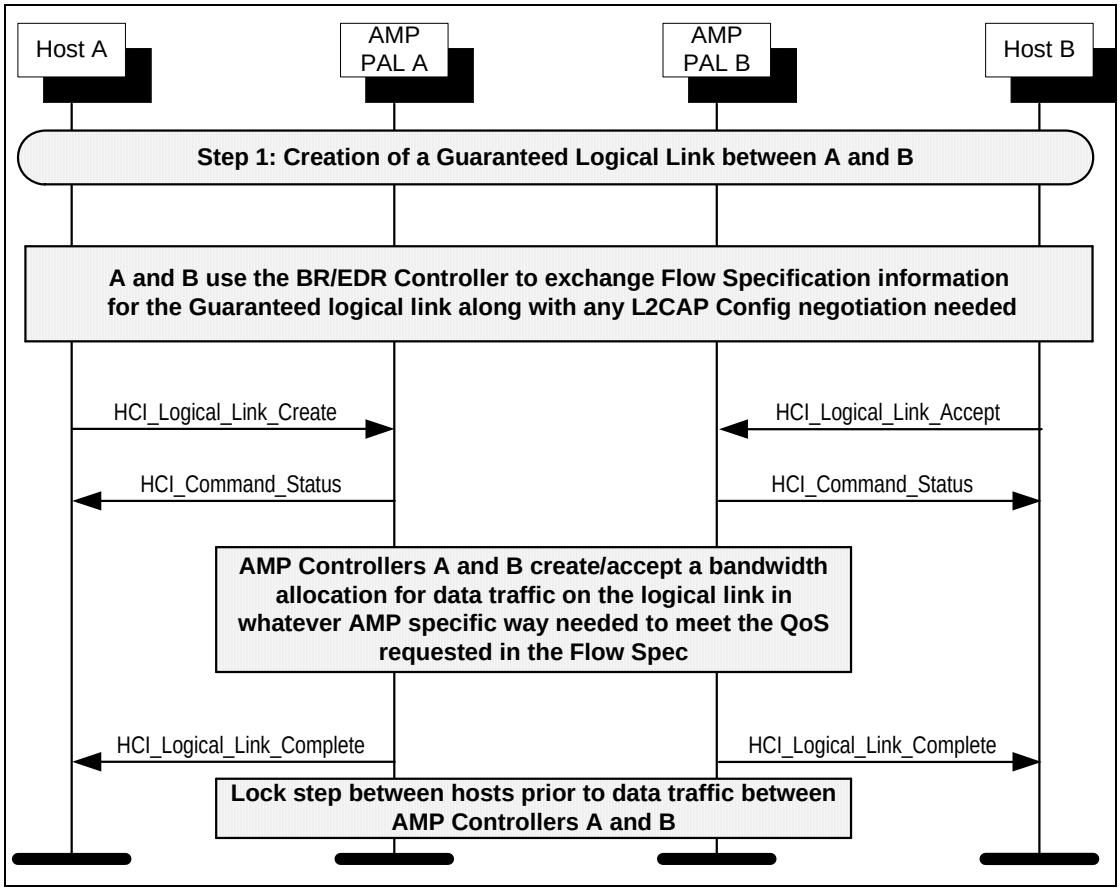



*Figure 4.46: Guaranteed Logical Link Creation*

**Step 2:** Host A sends a Logical Link Disconnect command to its AMP Controller. AMP Controllers A and B do whatever AMP specific action is required to meet the request. Each AMP Controller sends the Logical Link Disconnect Complete event to its host.

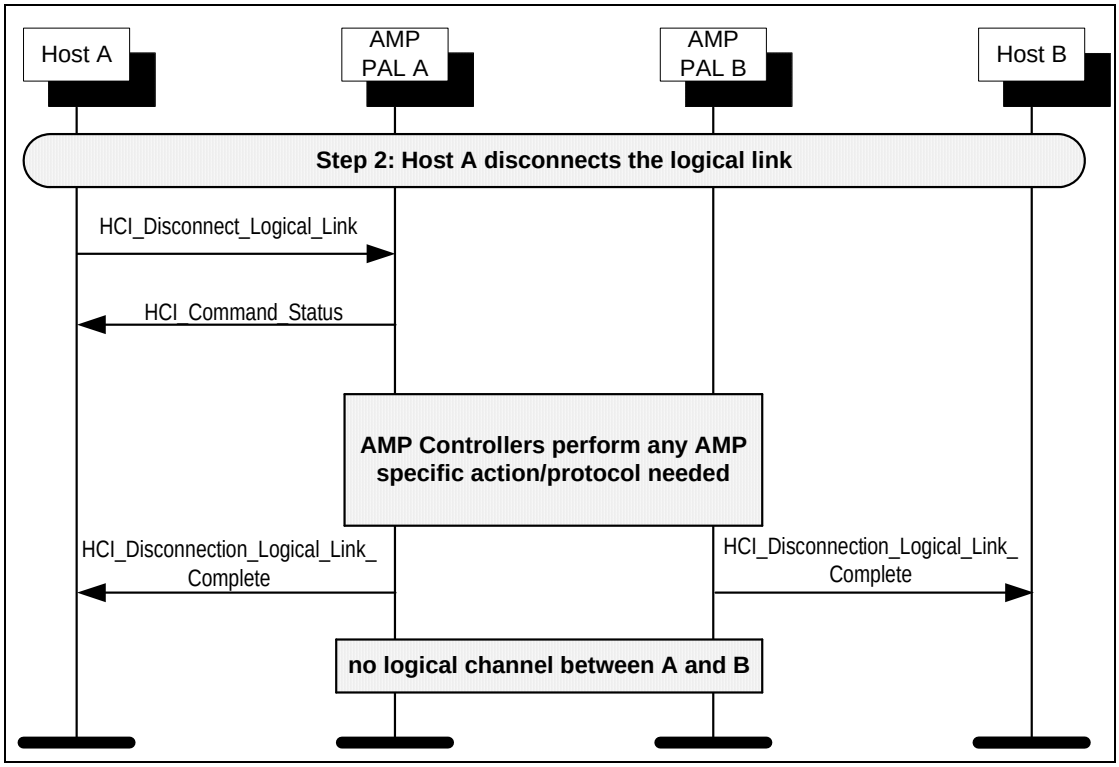



*Figure 4.47: Host A disconnects the logical link*

**Step 3:** During normal operation the Host may modify logical link characteristics.

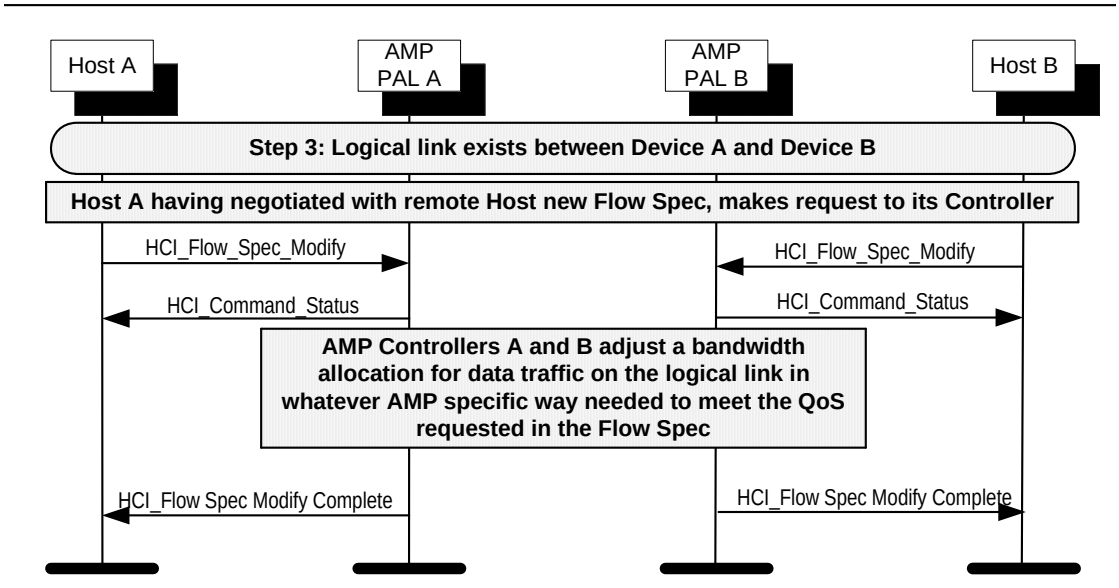



*Figure 4.48: Host A modifies Flow_Spec on an existing logical link*

## 4.17 AMP TEST MODE SEQUENCE CHARTS

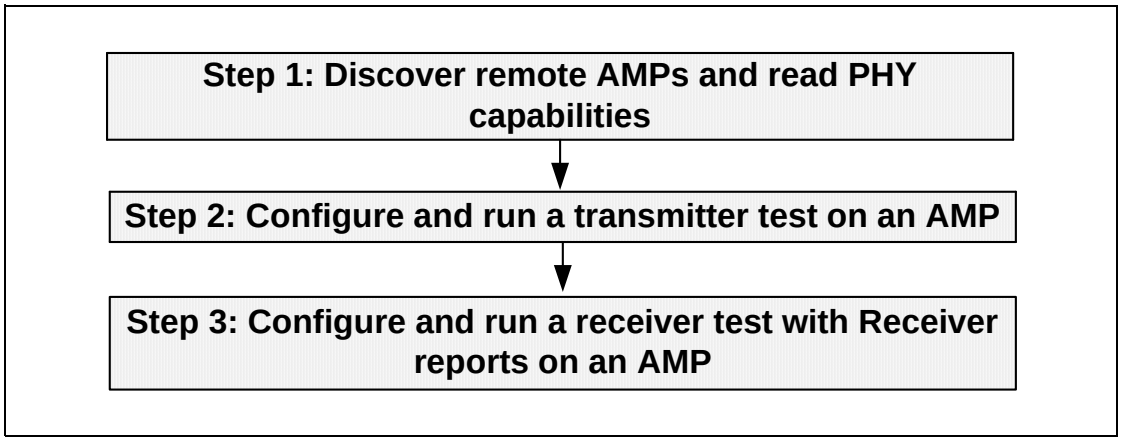



*Figure 4.49: Overview diagram for AMP test mode*

### 4.17.1 Discover the AMP Present and Running Transmitter and Receiver Tests

The process of discovering the number of AMPs and running transmitter and receiver tests is detailed in the following three steps.

**Step 1**: The tester sends an AMP Discovery Request to the AMP Test Manager of the DUT over the BR/EDR ACL connection. The AMP Test Manager will reply with the AMP Discover Response to the tester with a list of available AMP and type over the BR/EDR ACL connection. The tester then requests the PHY capabilities for each of the AMP it may test.

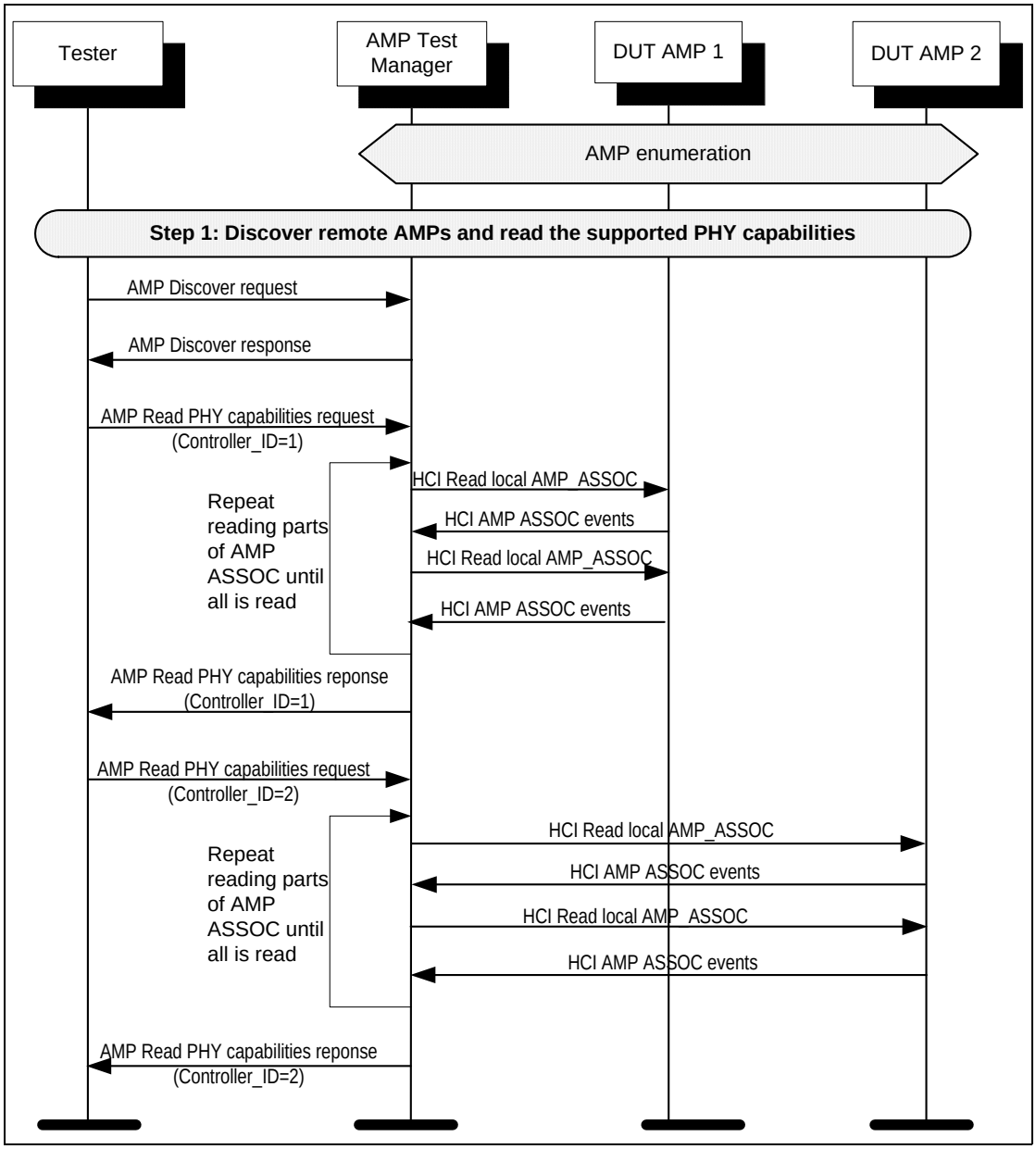



*Figure 4.50: Discover remote AMPs and their PHY capabilities*

**Step 2**: In order to configure a transmitter test, the Tester sends an HCI_AMP_Test_Command to the AMP Test Manager of the DUT using the L2CAP AMP Test Message for the AMP Controller indicated by the Controller_ID. The AMP Test Manager sends on the HCI_AMP_Test_Command on to the AMP indicated by the Controller_ID. When the HCI_AMP_Start_Test_Event is returned from the AMP controller, the event will be sent via the AMP Test Manager back to the Tester within an L2CAP AMP Test message.

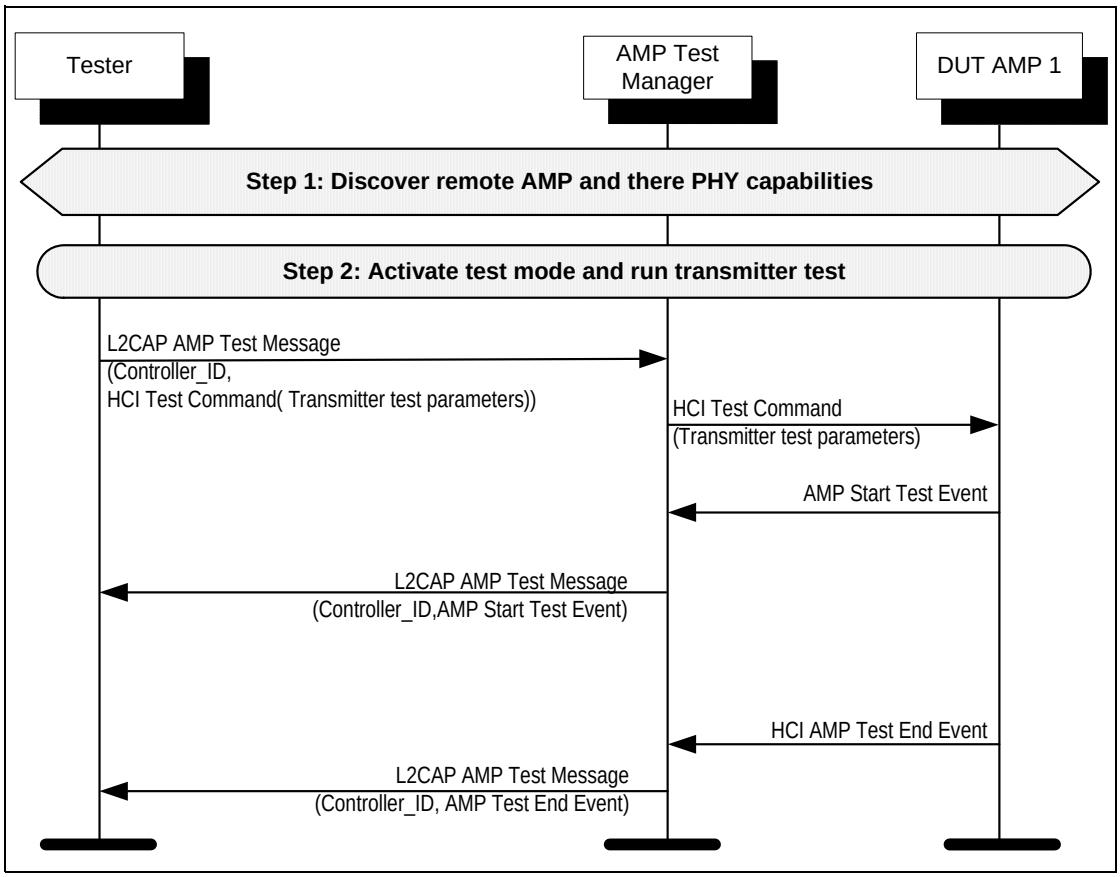



*Figure 4.51: Activate test mode and run a transmitter test*

**Step 3**: In order to Enable AMP Receiver Reports, the Tester sends an AMP test command to the AMP Test Manager of the DUT using the L2CAP AMP Test message for the AMP Controller indicated by the Controller_ID. The Command_Complete_Event is returned to the Tester via the AMP Test Manager in an L2CAP AMP Test Message.

In order to configure a receiver test, the Tester sends an HCI_AMP_Test_Command to the AMP Test Manager of the DUT using the L2CAP AMP Test Message for the AMP controller indicated by the Controller_ID. The AMP Test Manager sends on the HCI_AMP_Test_Command on to the AMP indicated by the Controller_ID. The HCI_AMP_Start_Event is returned to the Tester via the AMP Test Manager in an L2CAP AMP Test Message.

AMP_Receiver_Reports and the HCI_Test_End_Event are sent back to the Tester via the AMP Test Manager from the controller and then the Recever reports is disabled by the Tester using the HCI_Enable_AMP_Receiver_Reports command.

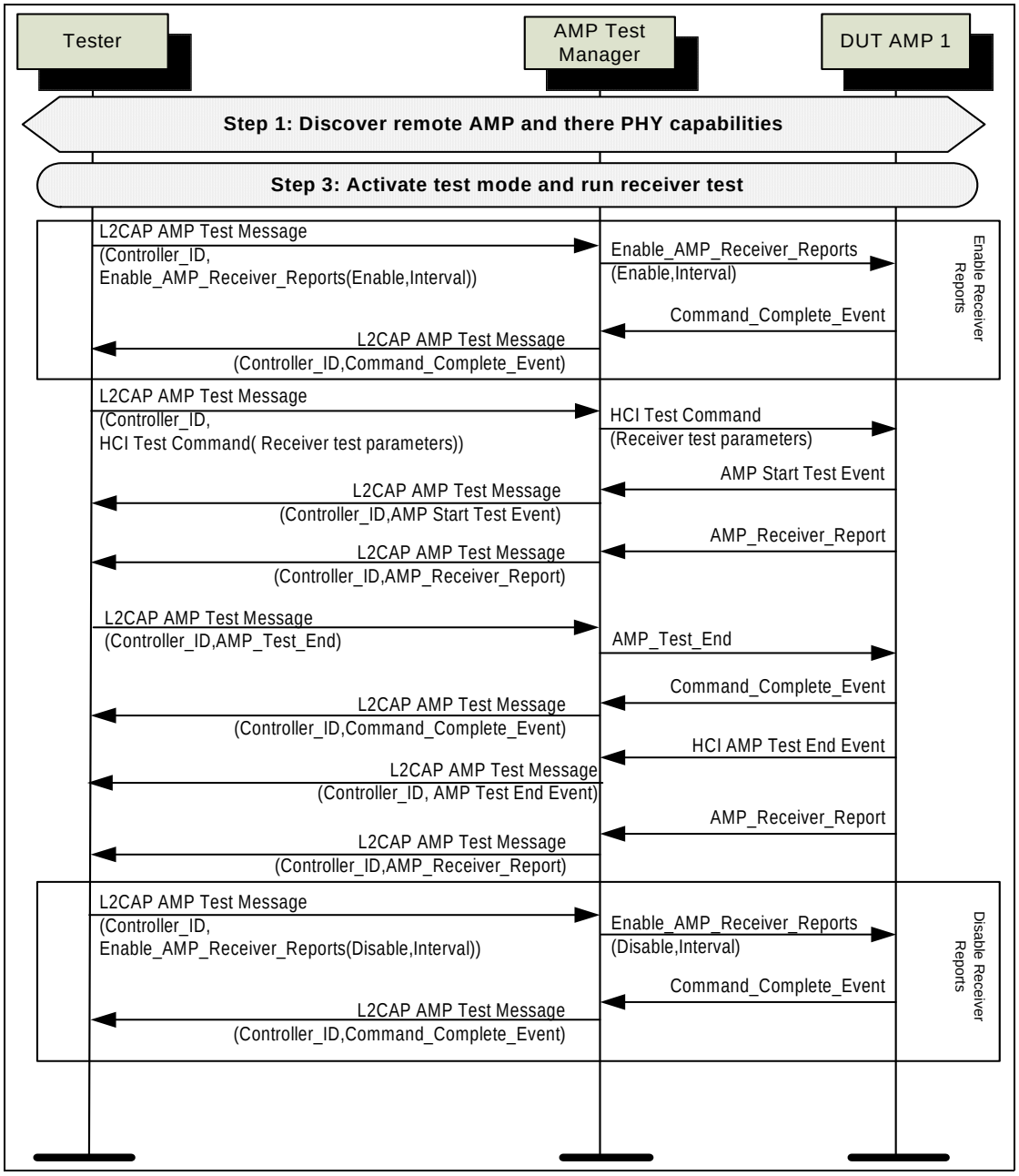



*Figure 4.52: Configure and run a receiver test with Receiver reports on an AMP*

# 5 SYNCHRONOUS CONNECTION ESTABLISHMENT AND DETACHMENT

## 5.1 SYNCHRONOUS CONNECTION SETUP

Using the HCI_Setup_Synchronous_Connection command, a host can add a synchronous logical channel to the link. A synchronous logical link can be provided by creating a SCO or an eSCO logical transport.

Note: An ACL Connection must be established before a synchronous connection can be created.

**Step 1a:** Master device requests a synchronous connection with a device. (See Figure 5.1 on page 929.)

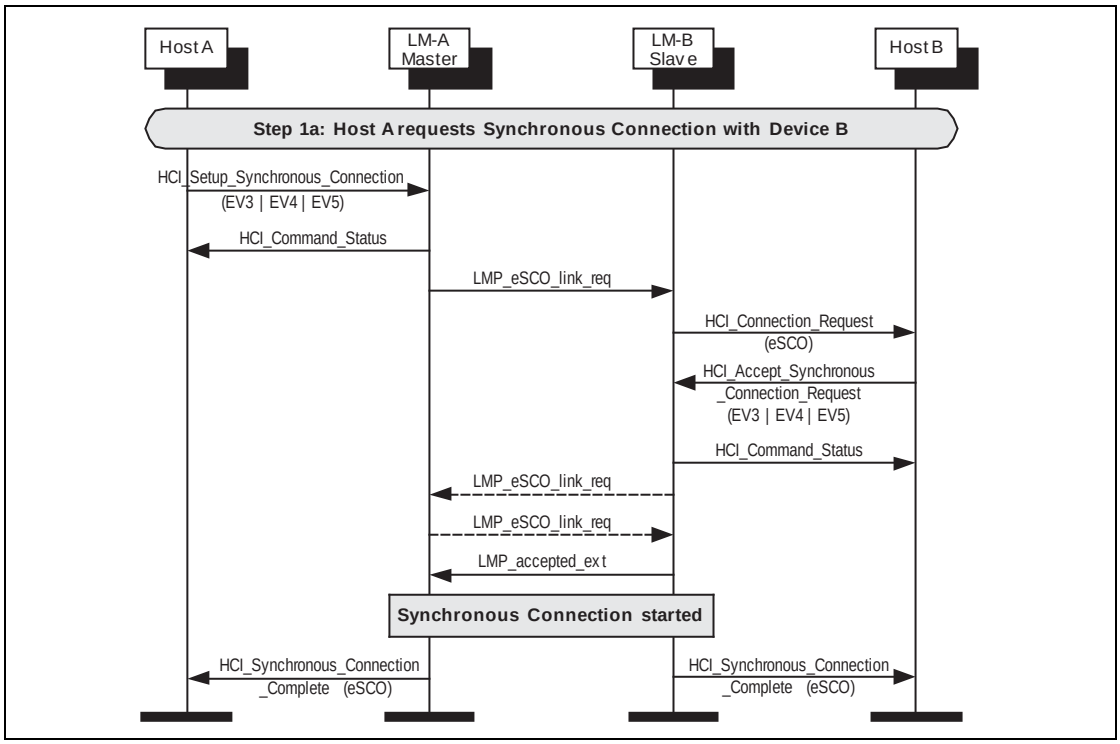



*Figure 5.1: Master requests synchronous EV3, EV4 OR EV5 connection.*

**Step 1b:** Slave device requests a synchronous connection with a device. (See Figure 5.2 on page 930.)

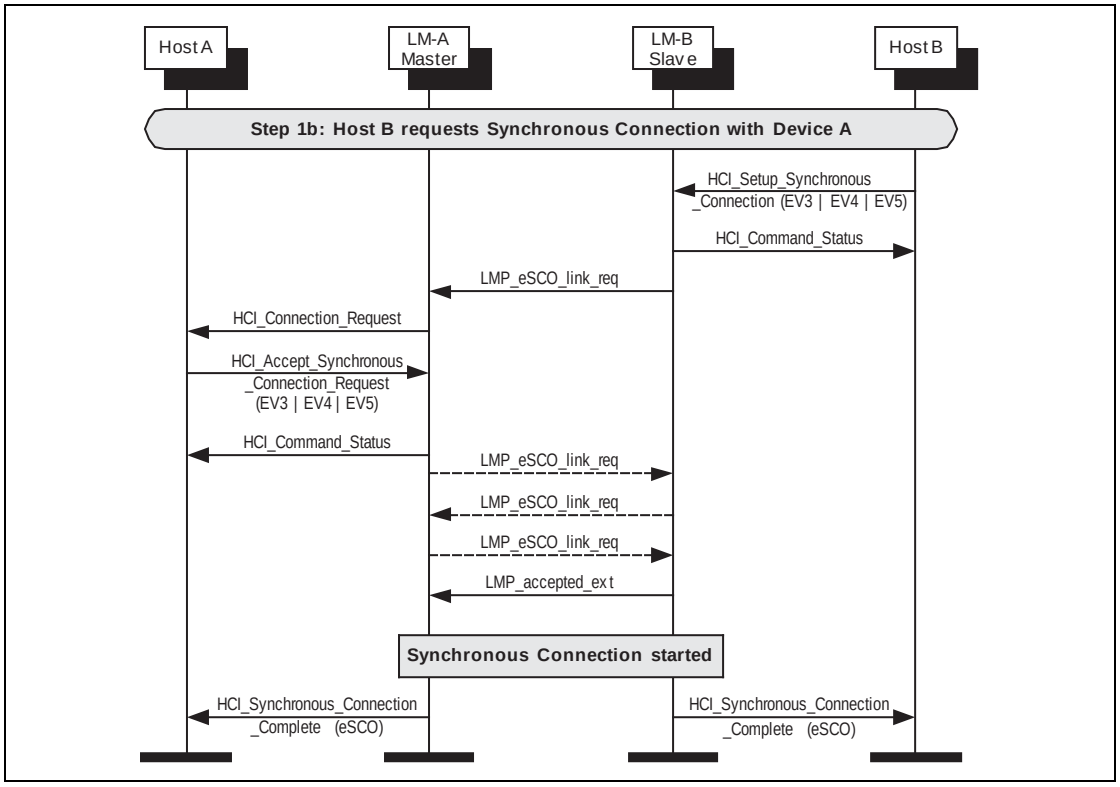



*Figure 5.2: Slave requests synchronous EV3, EV4 OR EV5 connection.*

**Step 1c:** Master device requests a SCO connection with a device. (See Figure 5.3 on page 930.)

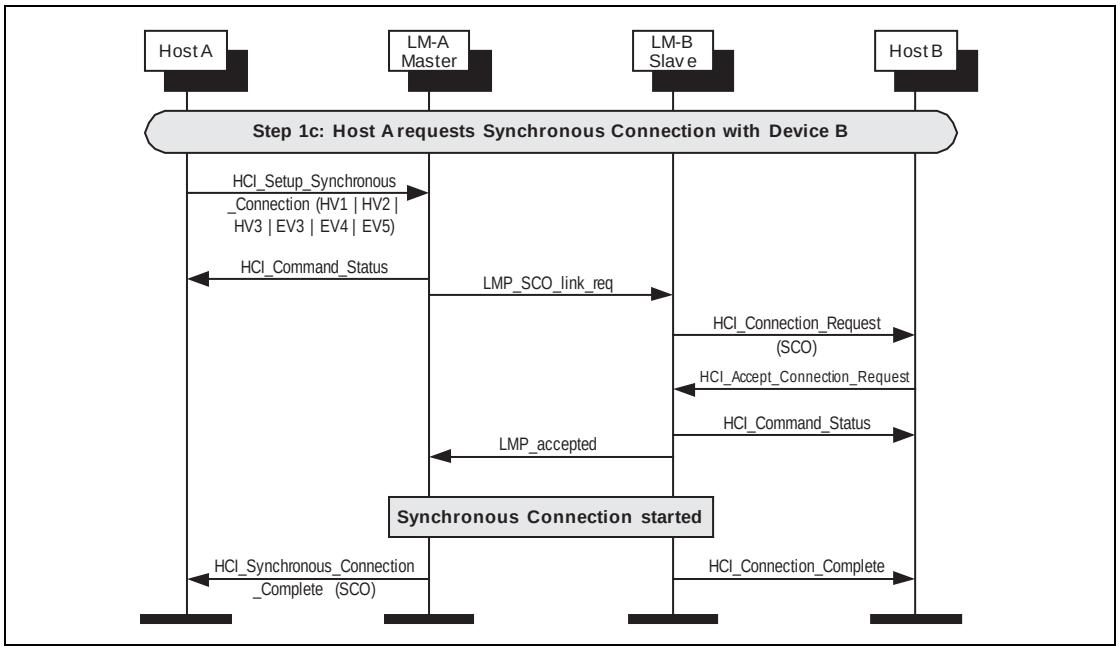



*Figure 5.3: Master requests synchronous connection using SCO.*

**Step 1d:** Master device requests a SCO connection with a device.
(See Figure 5.4 on page 931.)

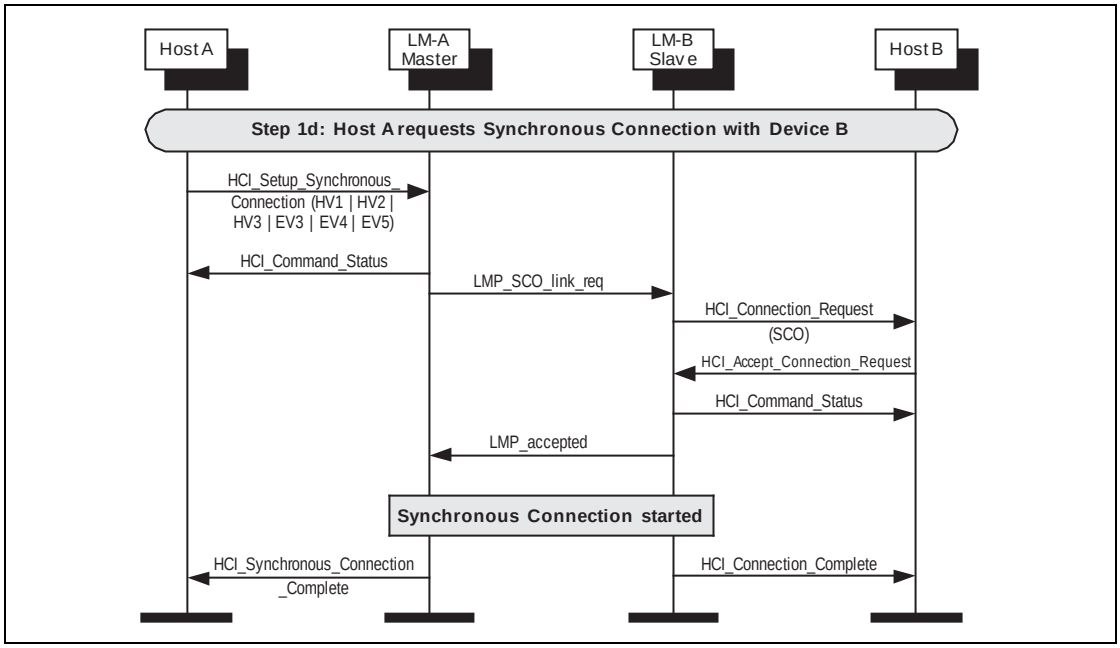



*Figure 5.4: Master requests synchronous connection with legacy slave.*

**Step 1e:** Host device requests a SCO connection with a device.
(See Figure 5.5 on page 931.)

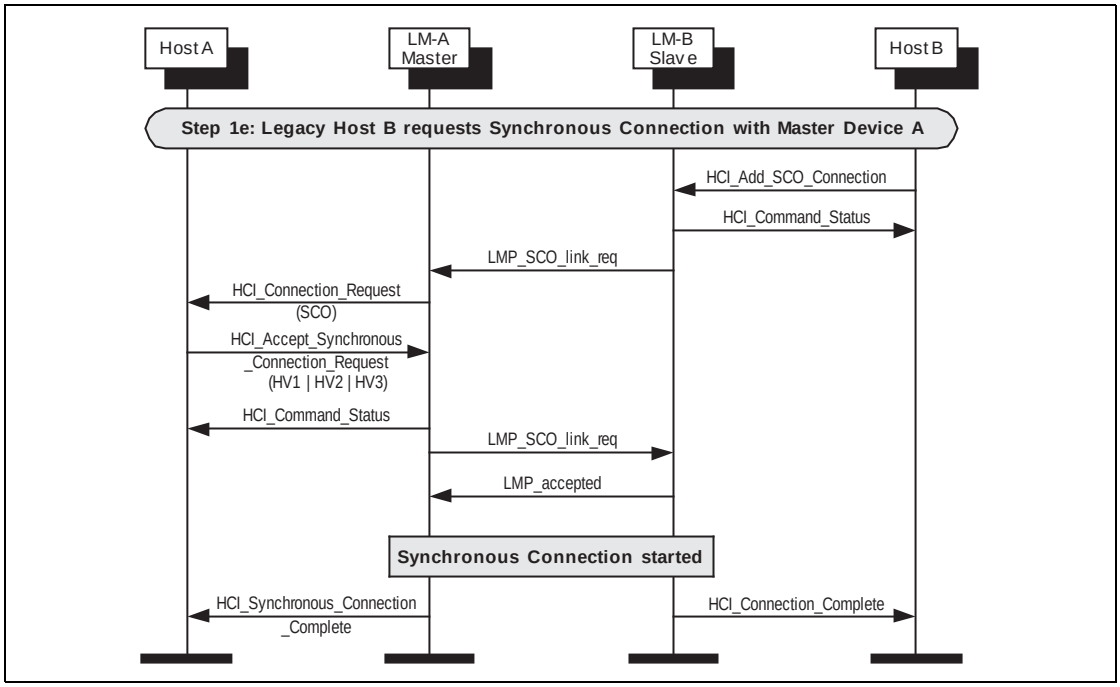



*Figure 5.5: Any device that supports only SCO connections requests a synchronous connection with a device.*

**Step 2a:** Master renegotiates eSCO connection (See Figure 5.6 on page 932.)

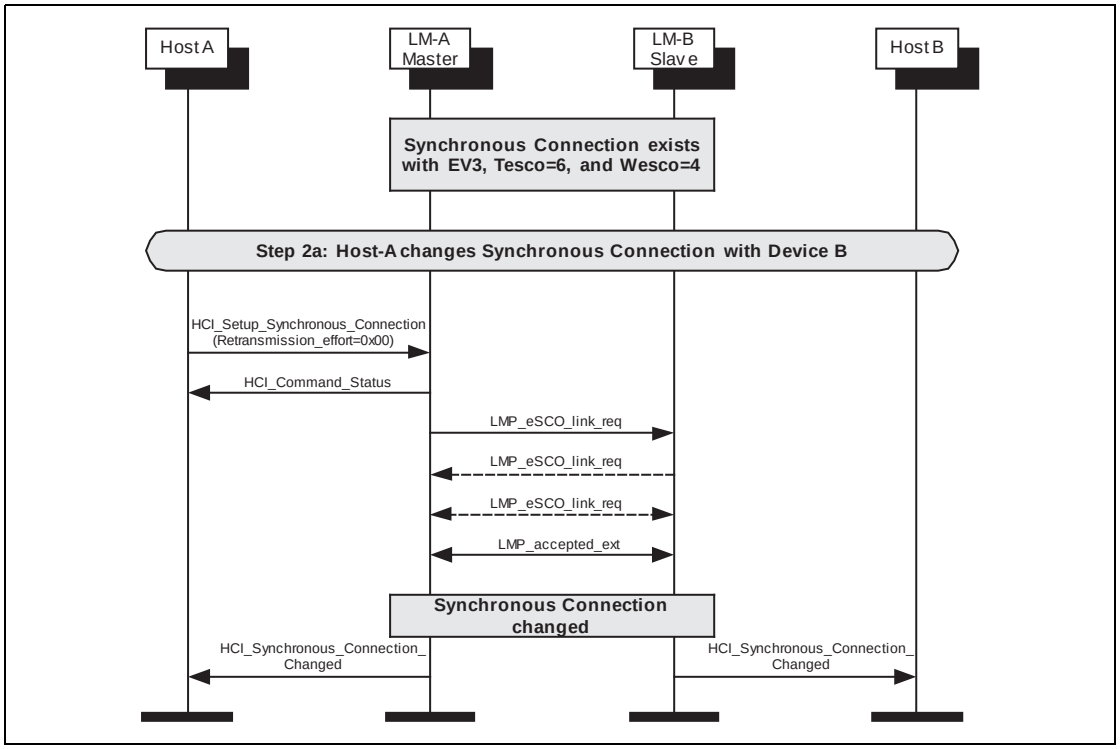



*Figure 5.6: Master renegotiates eSCO connection.*

**Step 2b:** Slave renegotiates eSCO connection (See Figure 5.7 on page 932.)

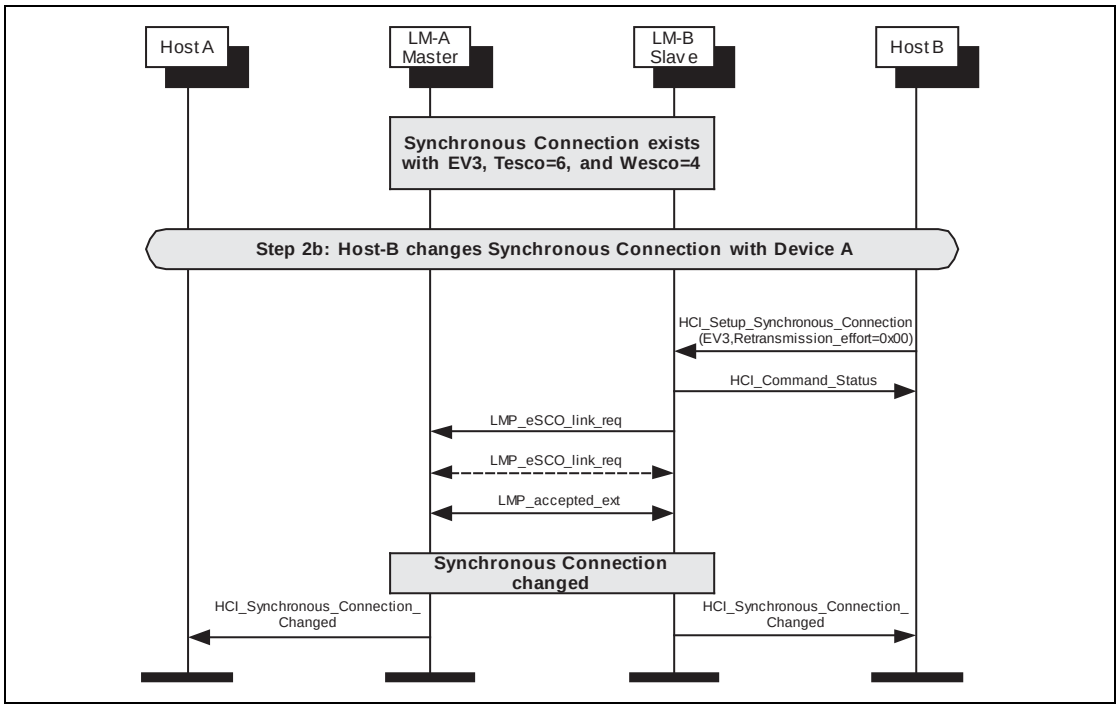



*Figure 5.7: Slave renegotiates eSCO connection.*

**Step 3a:** eSCO Disconnection. (See Figure 5.8 on page 933.)

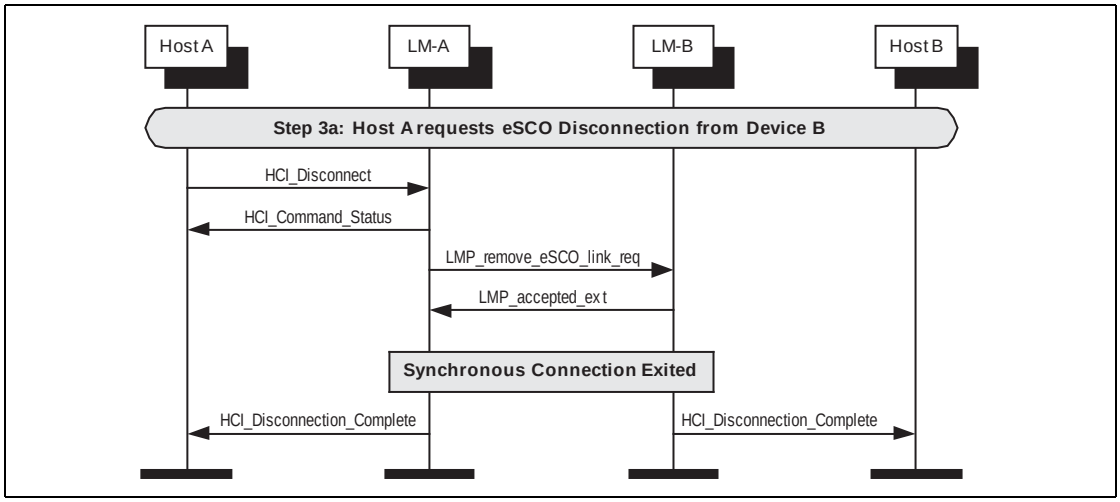



*Figure 5.8: Synchronous disconnection of eSCO connection.*

**Step 3b:** SCO Disconnection. (See Figure 5.9 on page 933.)

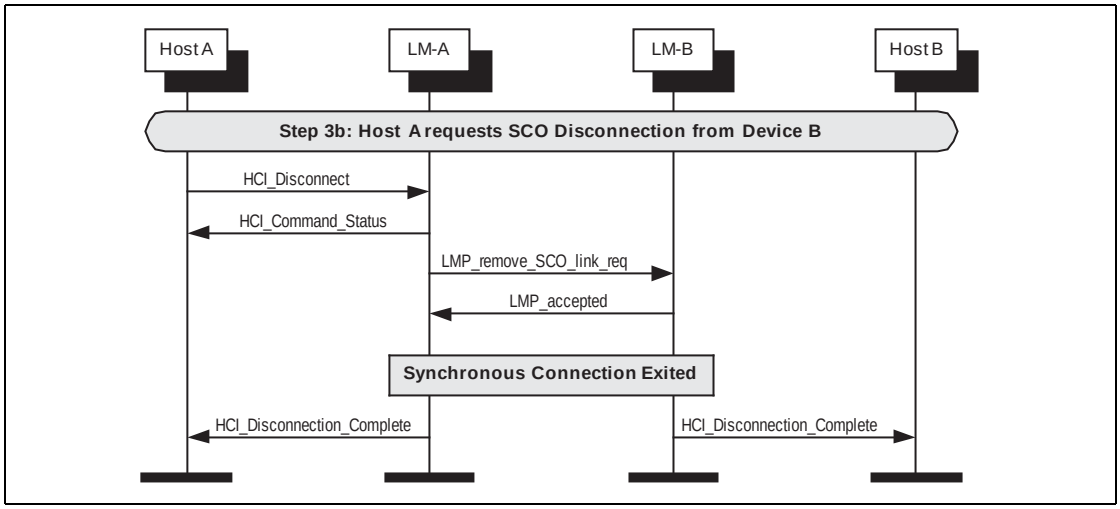



*Figure 5.9: Synchronous disconnection of SCO connection.*

# 6 SNIFF, HOLD AND PARK

Entry into Sniff mode, Hold mode or Park state requires an established ACL Connection.

## 6.1 SNIFF MODE

The HCI_Sniff_Mode command is used to enter sniff mode. The HCI_Exit_Sniff_Mode command is used to exit sniff mode.

**Step 1:** Host requests to enter sniff mode. Multiple LMP_sniff_req PDUs may be sent as the parameters for sniff mode are negotiated. (See Figure 6.1 on page 934.)

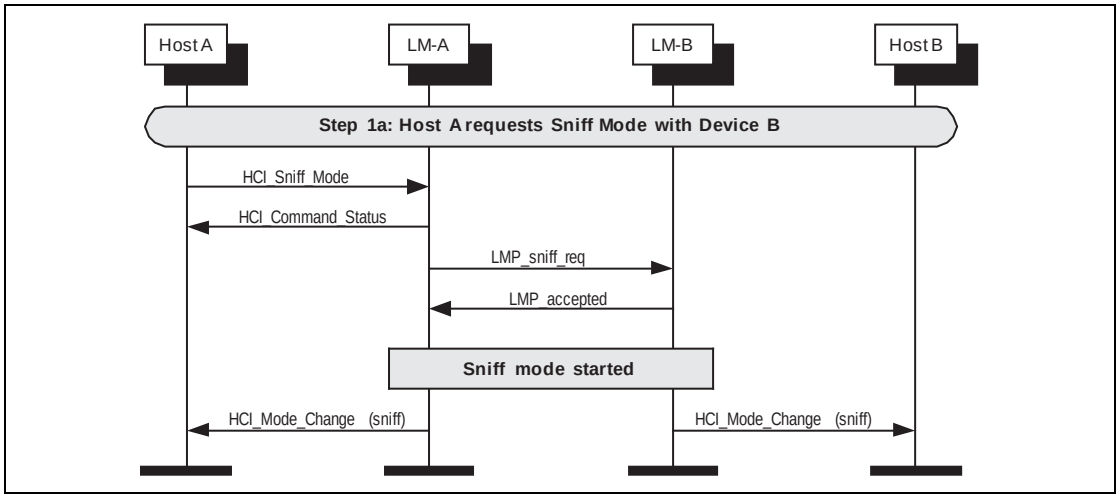



*Figure 6.1: Sniff mode request.*

**Step 2:** Host requests to exit sniff mode. (See Figure 6.2 on page 934.)

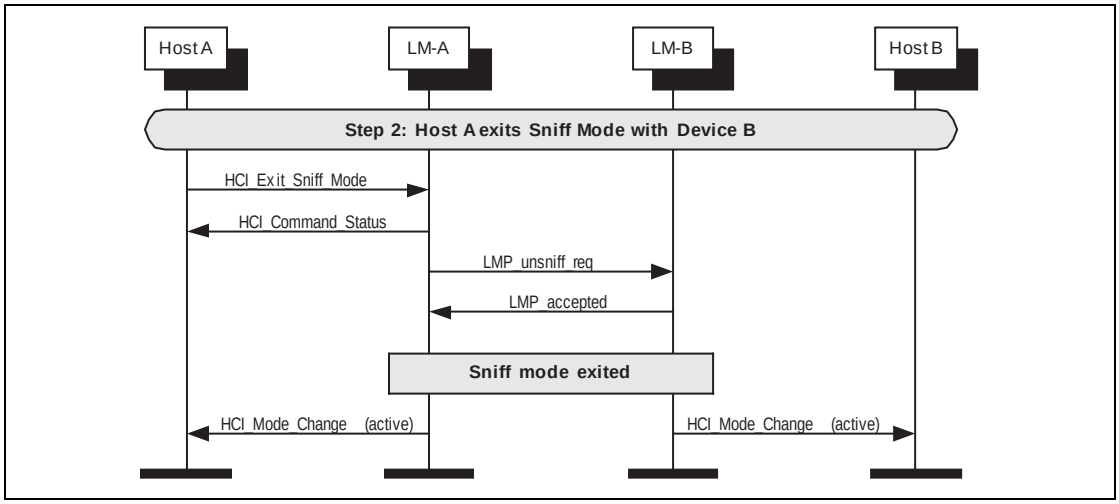



*Figure 6.2: Exit sniff mode request.*

## 6.2 HOLD MODE

The HCI_Hold_Mode command can be used to place a device into hold mode. The Controller may do this by either negotiating the hold mode parameters or forcing hold mode. Hold mode will automatically end after the negotiated length of time.

**Step 1a:** A host requests hold mode. (See Figure 6.3 on page 935.)

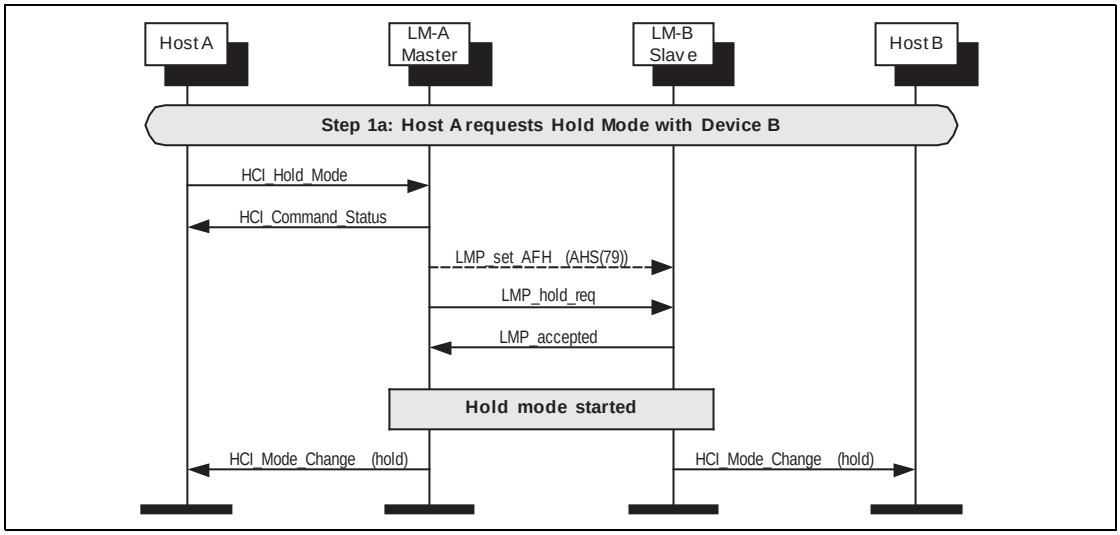



*Figure 6.3: Hold request.*

**Step 1b:** A host may force hold mode. (See Figure 6.4 on page 935.)

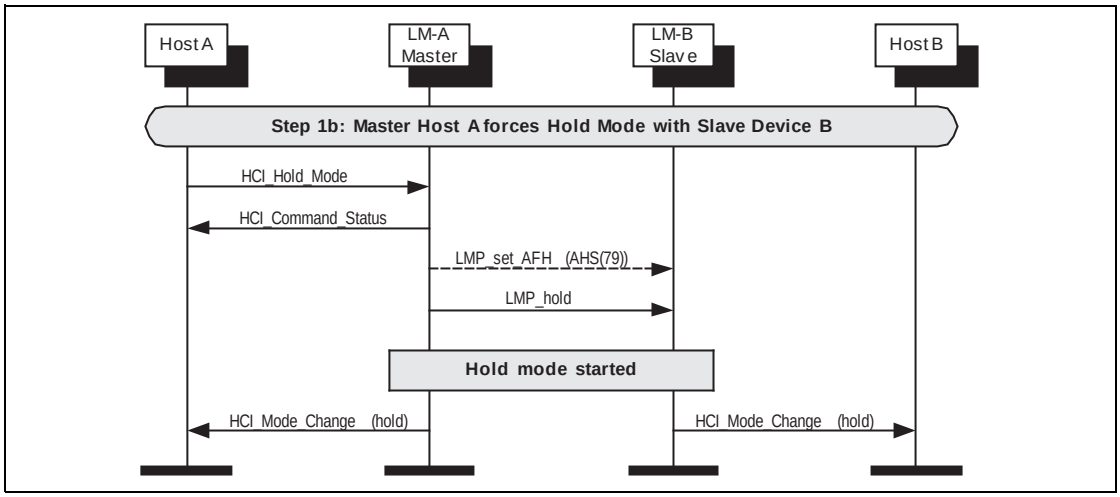



*Figure 6.4: Master forces hold mode.*

**Step 1c:** A slave device requests hold mode. (See Figure 6.5 on page 936.)

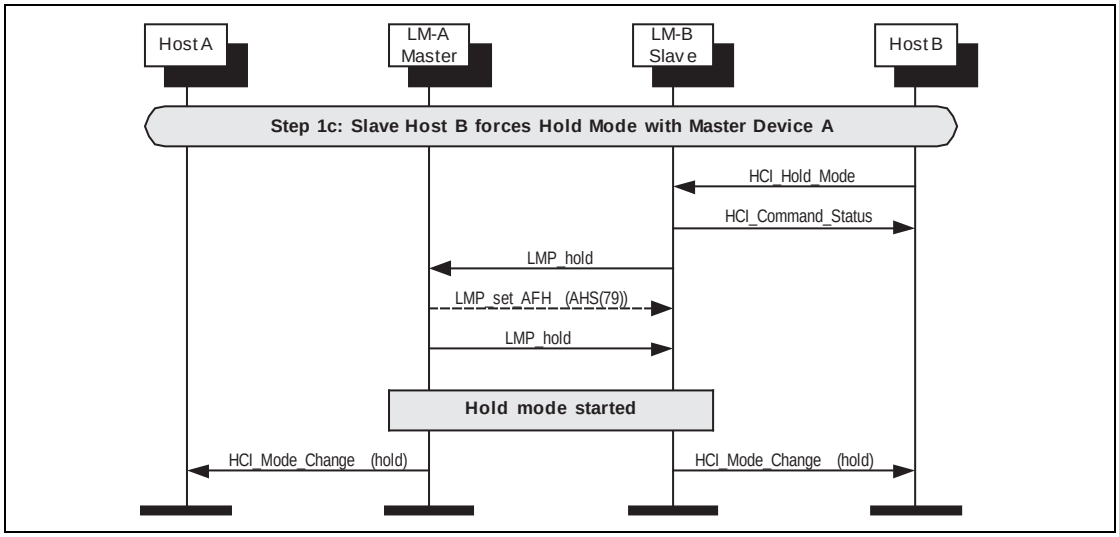



*Figure 6.5: Slave forces hold mode.*

**Step 2:** When hold mode completes the hosts are notified using the HCI_Mode_Change event. (See Figure 6.6 on page 936.)

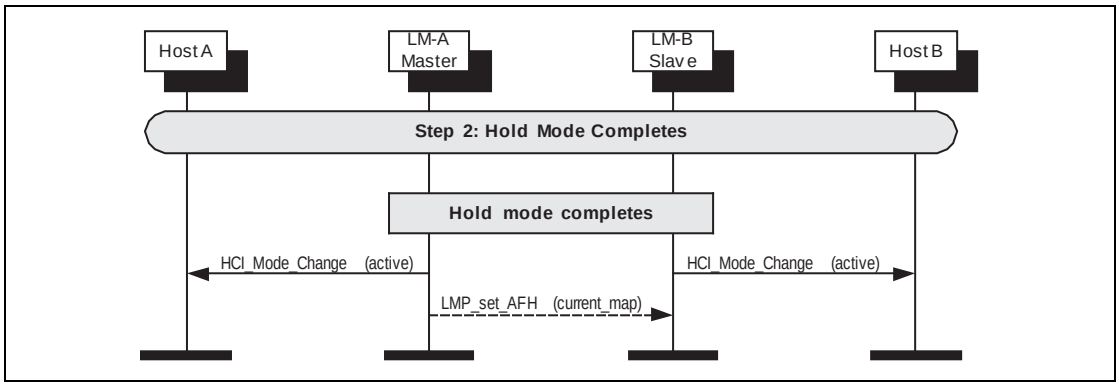



*Figure 6.6: Hold mode completes.*

## 6.3 PARK STATE

Park state can be entered by using the HCI_Park_State command.

**Step 1a:** The master requests to place the slave in the park state. Before sending the LMP_park_req PDU, the master may disable AFH by setting the connection into AHS(79). (See Figure 6.7 on page 937.)

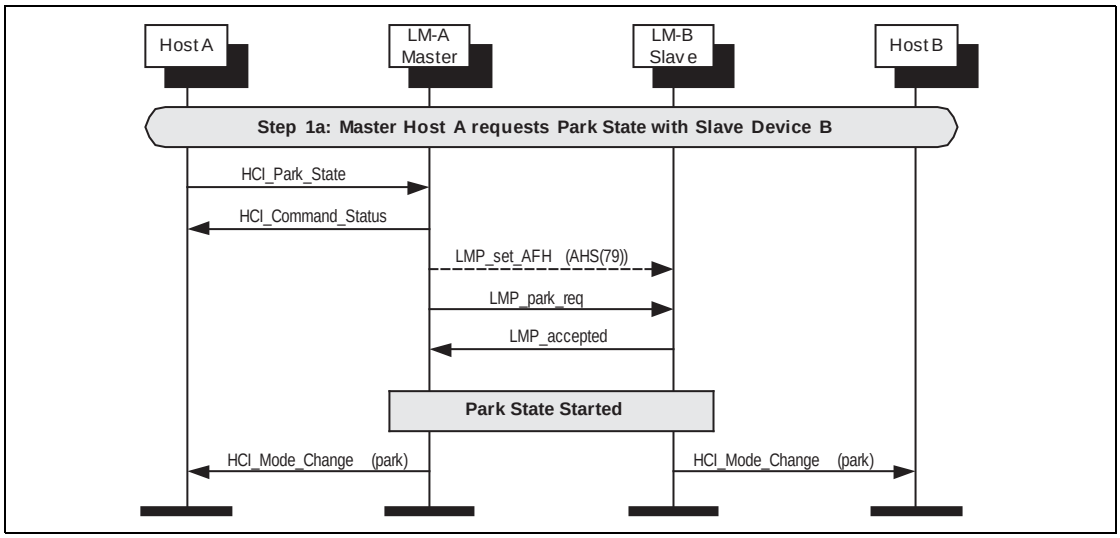



*Figure 6.7: Park state request from master.*

**Step 1b:** The slave requests to be placed in the park state. Before sending the LMP_park_req PDU back to the slave, the master may disable AFH by setting the connection into AHS(79). (See Figure 6.8 on page 937.)

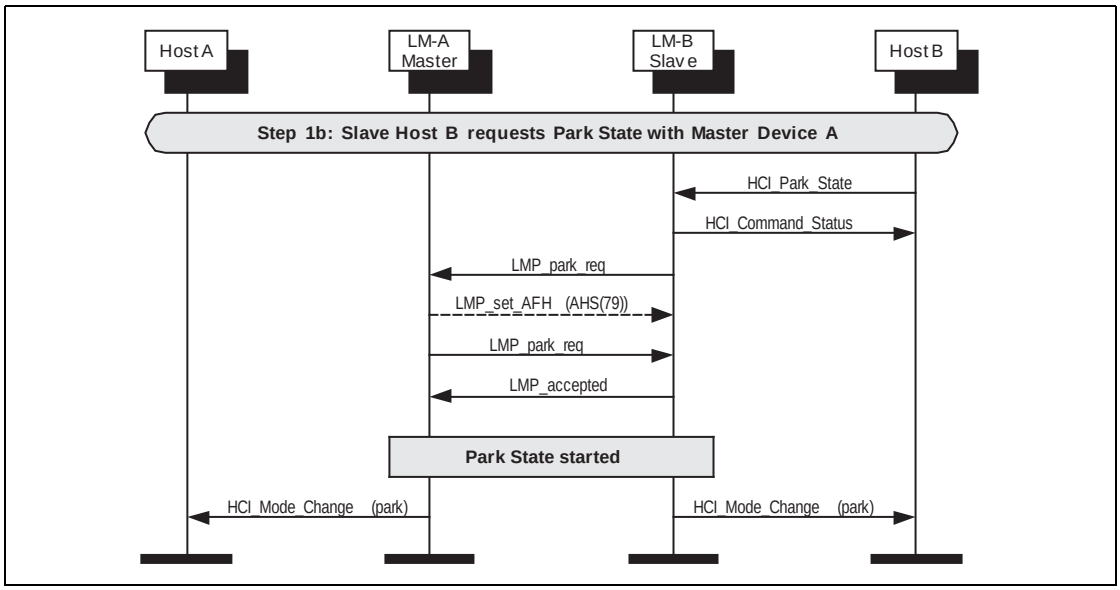



*Figure 6.8: Park state request from slave.*

**Step 2:** When in the park state, a slave still needs to unparked for link supervision purposes. The master sends an LMP_unpark_PM_ADDR_req PDU or an LMP_unpark_BD_ADDR_req PDU to the slave during the beacon. Only the PM_ADDR version is illustrated in the figure. (See Figure 6.9 on page 938.)

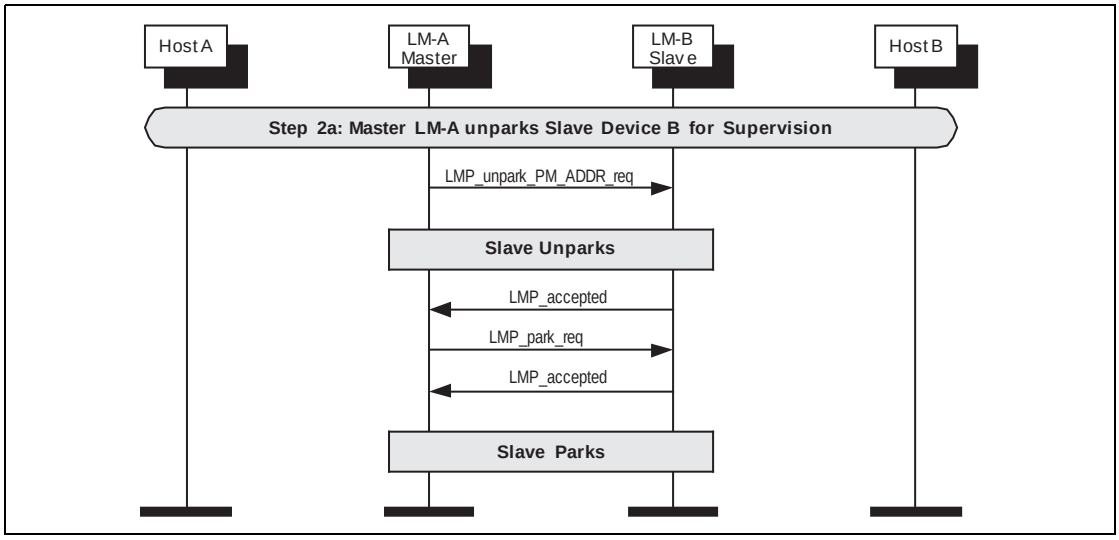



*Figure 6.9: Master unparks slave for supervision.*

**Step 3a:** A master may unpark a slave to exit park state. The master should re-enable AFH by setting the current AFH channel map to the unparked slave. (See Figure 6.10 on page 938.)

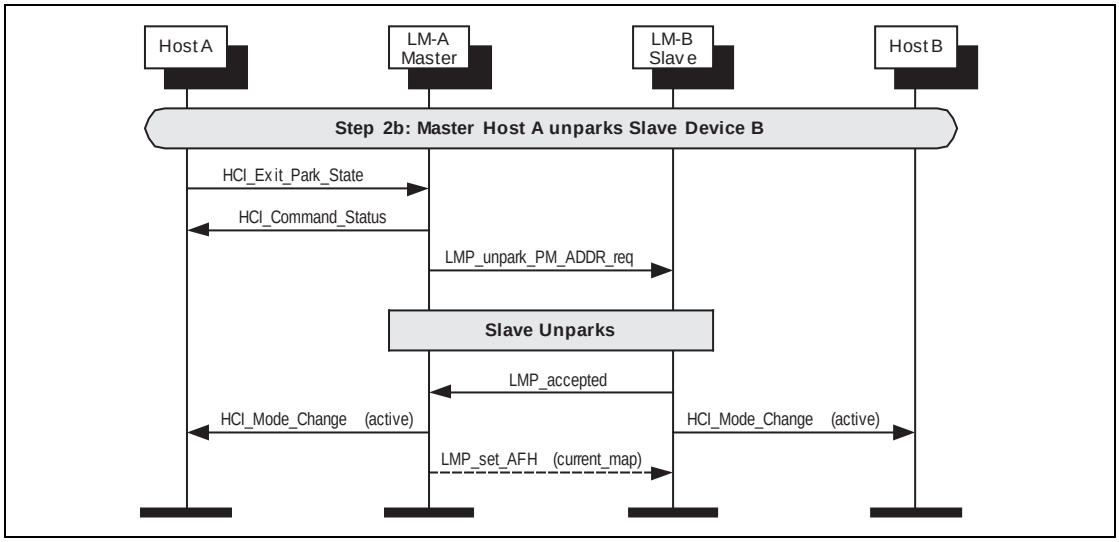



*Figure 6.10: Master exits park state with slave.*

**Step 3b:** A slave may request to be unparked by sending a message in an access window. It will then receive instructions from the master to unpark. The master should re-enable AFH by setting the current AFH channel map to the unparked slave. (See Figure 6.11 on page 939.)

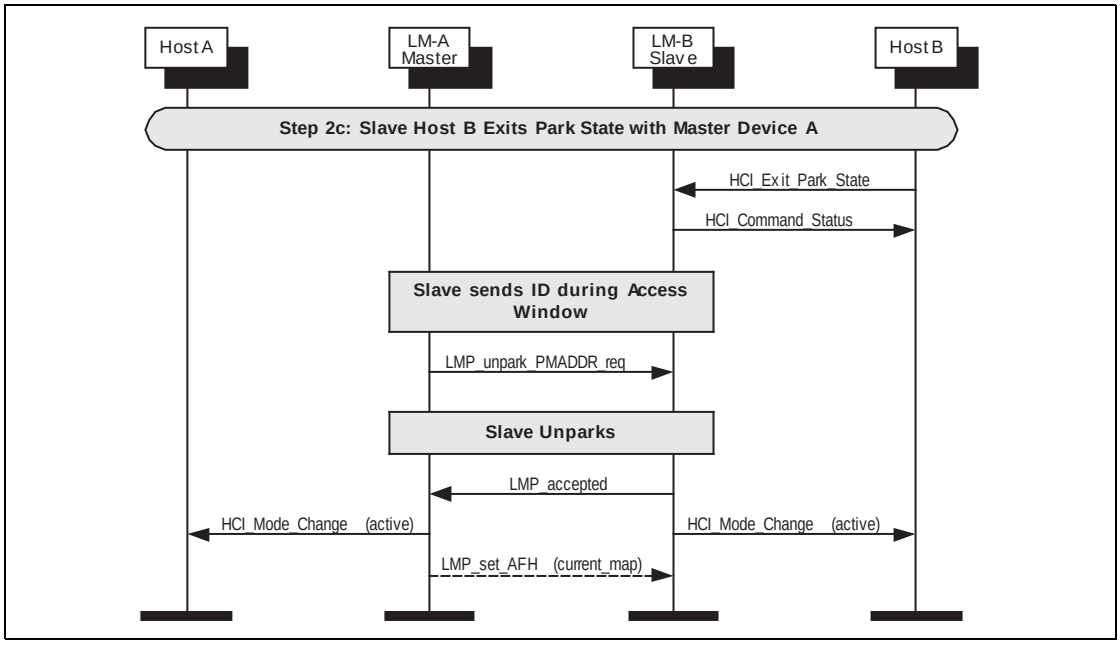



*Figure 6.11: Slave exits park state with master.*

# 7 BUFFER MANAGEMENT, FLOW CONTROL

Buffer management is very important for resource limited devices. This can be achieved on the Host Controller Interface using the HCI_Read_Buffer_Size command, and the HCI_Number_Of_Completed_Packets event, and the HCI_Set_Host_Controller_To_Host_Flow_Control, HCI_Host_Buffer_Size and HCI_Host_Number_Of_Completed_Packets commands.

**Step 1:** During initialization, the host reads the buffer sizes available in the Controller. When an HCI Data Packet has been transferred to the remote device, and a Baseband acknowledgement has been received for this data, then an HCI_Number_Of_Completed_Packets event will be generated. (See Figure 7.1 on page 940.)

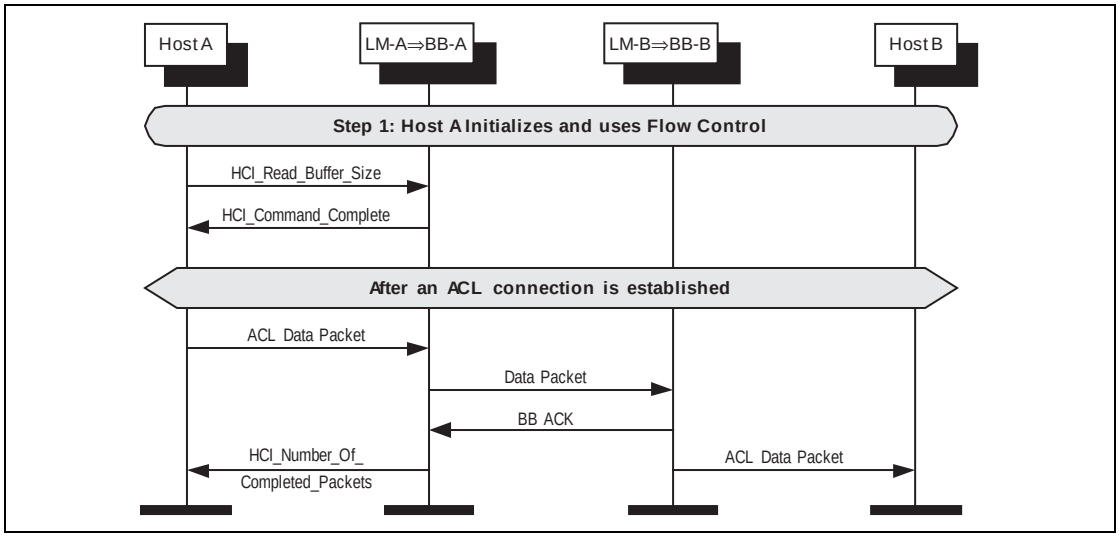



*Figure 7.1: Host to Controller flow control.*

**Step 2:** During initialization, the host notifies the Controller that host flow control shall be used, and then the host buffer sizes available. When a data packet has been received from a remote device, an HCI Data Packet is sent to the host from the Controller, and the host shall acknowledge its receipt by sending HCI_Host_Number_Of_Completed_Packets. (See Figure 7.2 on page 941.)

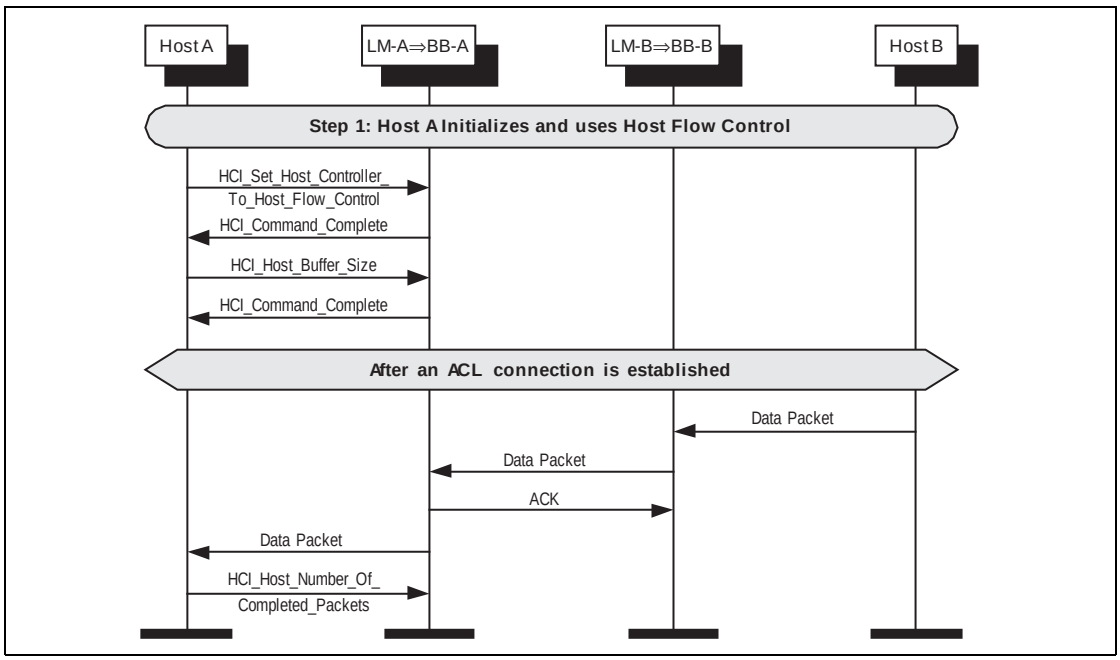



*Figure 7.2: Controller to Host flow control.*

# 8 LOOPBACK MODE

The loopback modes are used for testing of a device only.

## 8.1 LOCAL LOOPBACK MODE

The local loopback mode is used to loopback received HCI Commands, and HCI ACL and HCI Synchronous packets sent from the Host to the Controller.

**Step 1:** The host enters local loopback mode. Four connection complete events are generated and then a command complete event.
(See Figure 8.1 on page 942.)

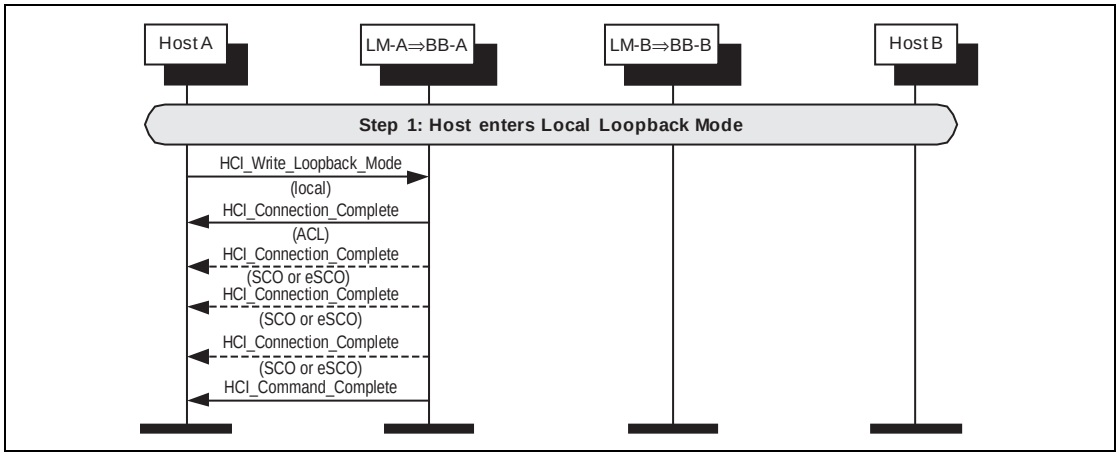



*Figure 8.1: Entering local loopback mode.*

**Step 2a:** The host sending HCI Data Packet will receive the exact same data back in HCI Data Packets from the Controller. (See Figure 8.2 on page 942.)

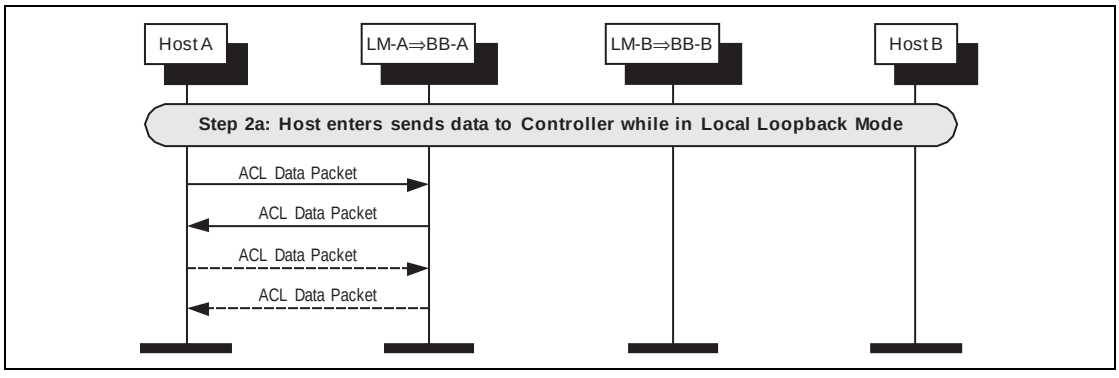



*Figure 8.2: Looping back data in local loopback mode.*

**Step 2b:** The host sending most HCI Command Packets to the Controller will receive an HCI_Loopback_Command event with the contents of the HCI Command Packet in the payload. (See Figure 8.3 on page 943.)

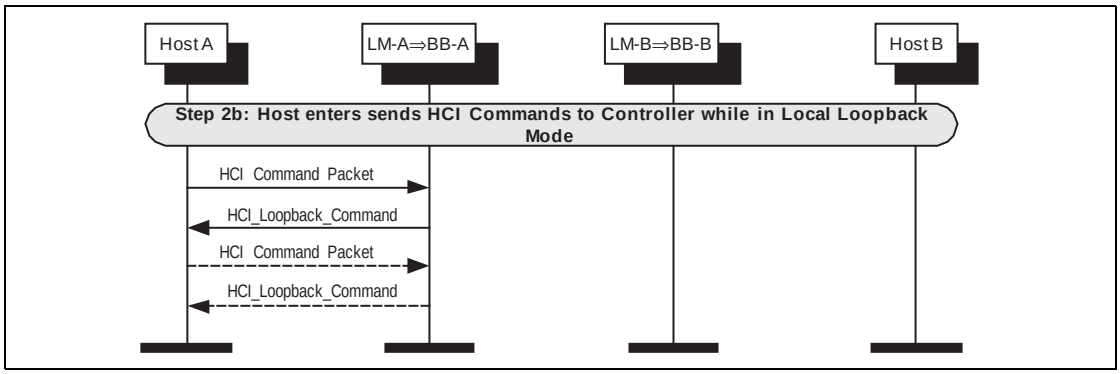



*Figure 8.3: Looping back commands in local loopback mode.*

**Step 3:** The host exits local loopback mode. Multiple disconnection complete events are generated before the command complete event.
(See Figure 8.4 on page 943.)

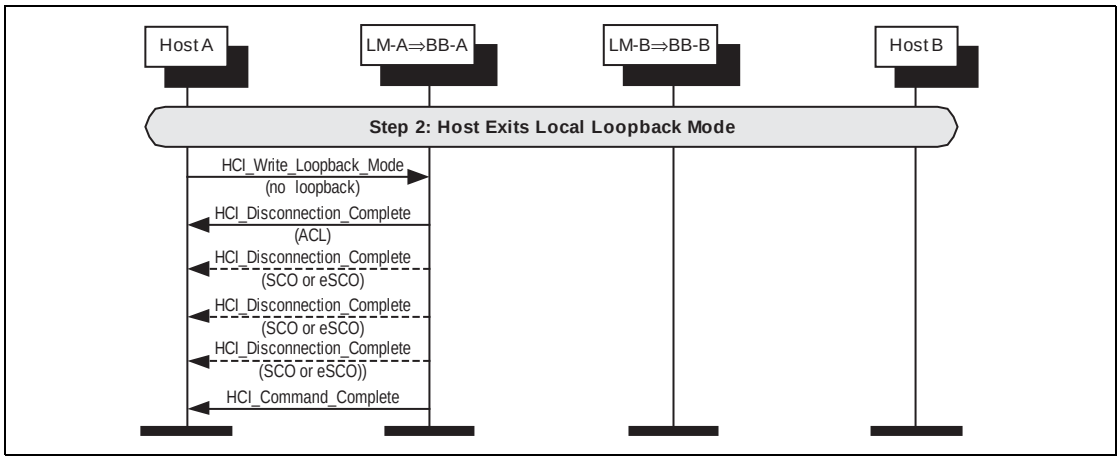



*Figure 8.4: Exiting local loopback mode.*

## 8.2 REMOTE LOOPBACK MODE

The remote loopback mode is used to loopback data to a remote device over the air.

**Step 1:** The remote host first sets up an connection to the local device. The local device then enables remote loopback. (See Figure 8.5 on page 944.)

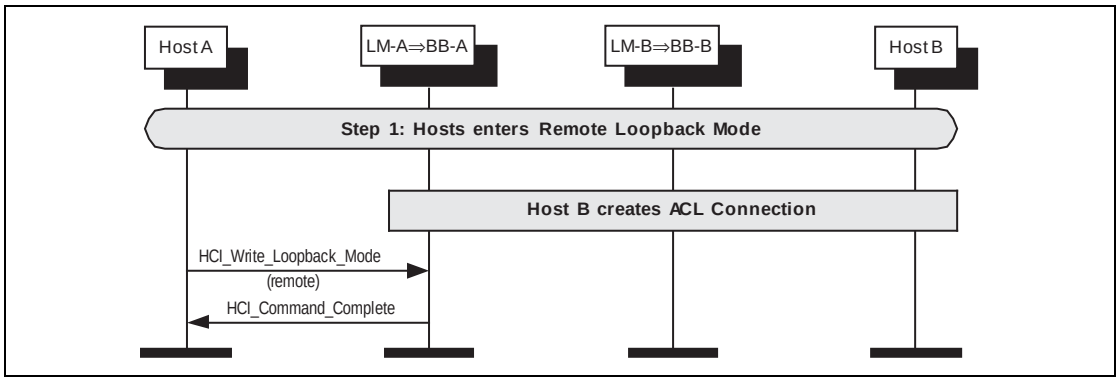



*Figure 8.5: Entering remote loopback mode.*

**Step 2:** Any data received from the remote host will be looped back in the Controller of the local device. (See Figure 8.6 on page 944.)

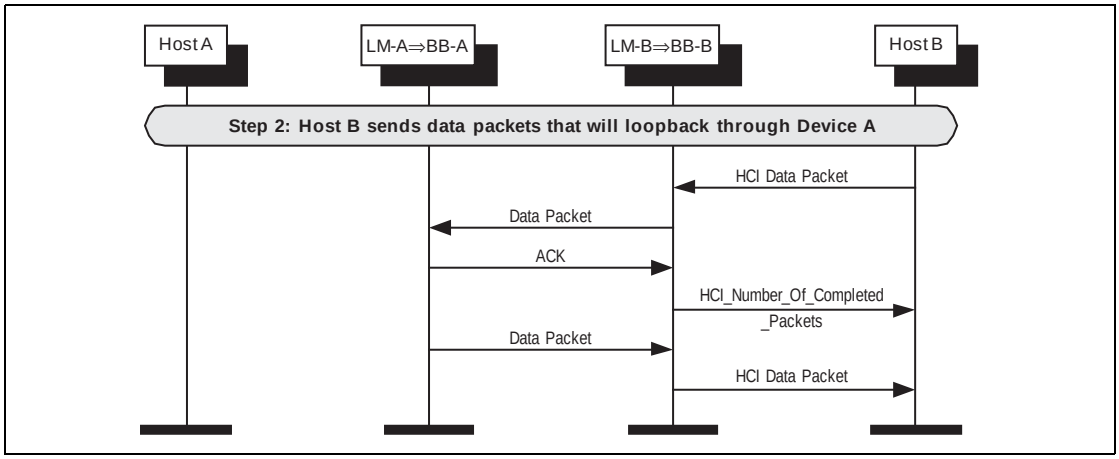



*Figure 8.6: Looping back data in Remote Loopback Mode.*

**Step 3:** The local host exits remote loopback mode. Any connections can then be disconnected by the remote device. (See Figure 8.7 on page 945.)

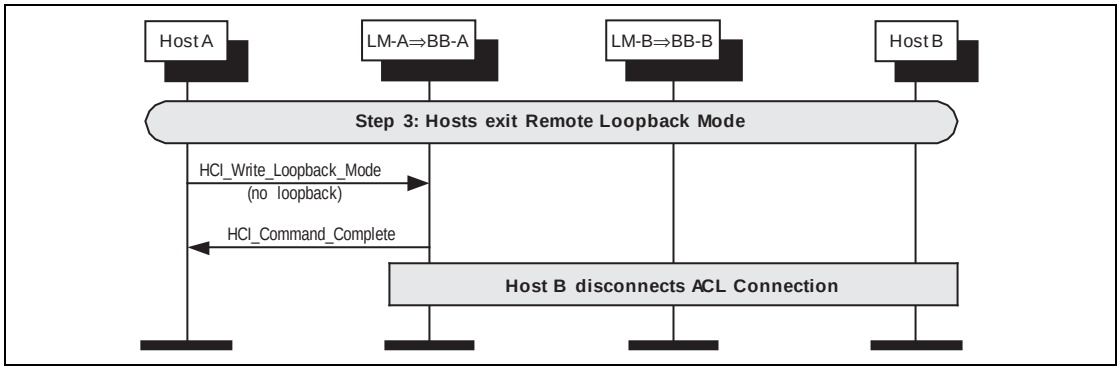



*Figure 8.7: Exiting remote loopback mode.*

# 9 LIST OF FIGURES

Figure 1.1: Example MSC. ..... 740
Figure 2.1: Remote name request. ..... 741
Figure 2.2: Remote name request if no current baseband connection. ..... 742
Figure 2.3: Remote name request with baseband connection. ..... 743
Figure 2.4: Host A starts inquiry procedure. ..... 743
Figure 2.5: LM-A performs inquiry and reports result. ..... 744
Figure 2.6: Host A cancels inquiry. ..... 744
Figure 2.7: LM-A terminates current inquiry. ..... 745
Figure 2.8: Host A starts periodic inquiry. ..... 745
Figure 2.9: LM-A periodically performs an inquiry and reports result. ..... 746
Figure 2.10: LM-A terminates current inquiry. ..... 746
Figure 2.11: Host A decides to exit periodic inquiry. ..... 746
Figure 3.1: Overview diagram for connection setup. ..... 747
Figure 3.2: Host A requests connection with device B. ..... 748
Figure 3.3: LM-A and LM-B exchange features. ..... 748
Figure 3.4: LM-A requests host connection. ..... 748
Figure 3.5: Device B rejects connection request. ..... 749
Figure 3.6: Device B accepts connection request. ..... 749
Figure 3.7: Device B accepts connection requests as master. ..... 750
Figure 3.8: LM-A starts adaptive frequency hopping. ..... 750
Figure 3.9: Authentication initiated. ..... 751
Figure 3.10: Pairing during connection setup. ..... 752
Figure 3.11: Authentication during connection setup. ..... 753
Figure 3.12: Starting encryption during connection setup. ..... 753
Figure 3.13: LM-A and LM-B finishes connection setup. ..... 754
Figure 3.14: Host A decides to disconnect. ..... 754
Figure 4.1: Authentication requested. ..... 755
Figure 4.2: Simple Pairing Flow Diagram ..... 756
Figure 4.3: Optional OOB Information Collection ..... 757
Figure 4.4: Enable Simple Pairing ..... 757
Figure 4.5: L2CAP Connection Request for a Secure Service ..... 758
Figure 4.6: Optional OOB Information Transfer ..... 758
Figure 4.7: Start Simple Pairing ..... 759
Figure 4.8: IO Capability Exchange ..... 759
Figure 4.9: Public Key Exchange ..... 760
Figure 4.10: Numeric Comparison Authentication ..... 761
Figure 4.11: Numeric Comparison Authentication (Failure on Initiating Side) ..... 762
Figure 4.12: Numeric Comparison Failure on Responding Side ..... 763
Figure 4.13: Passkey Entry Authentication ..... 764

Figure 4.14: Passkey Entry Failure on Responding Side .....765
Figure 4.15: Passkey Entry Failure on Initiating Side .....765
Figure 4.16: OOB Authentication .....766
Figure 4.17: OOB Authentication failure on initiating side .....766
Figure 4.18: DHKey Checks .....767
Figure 4.19: Calculate Link Key .....768
Figure 4.20: Start Encryption .....768
Figure 4.21: L2CAP Connection Response .....769
Figure 4.22: Link Supervision Timeout Event .....769
Figure 4.23: Encryption requested. .....770
Figure 4.24: Encryption off requested. .....770
Figure 4.25: Change connection link key. .....771
Figure 4.26: Change Connection Link Key with Encryption Pause Resume .....772
Figure 4.27: Change to master link key. .....773
Figure 4.28: Change to semi permanent link key. .....774
Figure 4.29: Read remote supported features. .....775
Figure 4.30: Read remote extended features. .....775
Figure 4.31: Read clock offset. .....776
Figure 4.32: Role Switch on an Encrypted link using encryption pause and resume. .....777
Figure 4.33: Refreshing encryption keys using encryption pause and resume .....778
Figure 4.34: Read remote version information. .....778
Figure 4.35: QoS flow specification. .....779
Figure 4.36: Master requests role switch. .....779
Figure 4.37: Slave requests role switch. .....780
Figure 4.38: Role switch is performed. .....780
Figure 4.39: Overview diagram for Physical Link creation and disconnection .....781
Figure 4.40: Get remote AMP data .....781
Figure 4.41: Host A tells its Controller to create the physical link .....782
Figure 4.42: Host A tells Host B to accept an incoming physical link .....782
Figure 4.43: A and B establish AMP physical link security .....783
Figure 4.44: Host A disconnects the physical link .....783
Figure 4.45: Controller reports to its Host of Link Loss .....784
Figure 4.46: Guaranteed Logical Link Creation .....785
Figure 4.47: Host A disconnects the logical link .....786
Figure 4.48: Host A modifies Flow_Spec on an existing logical link .....786
Figure 4.49: Overview diagram for AMP test mode .....787
Figure 4.50: Discover remote AMP and theire PHY capabilities .....788
Figure 4.51: Activate test mode and run a transmitter test .....789

Figure 4.52: Configure and run a receiver test with Receiver reports on an AMP ..... 790
Figure 5.1: Master requests synchronous EV3, EV4 OR EV5 connection. ..... 791
Figure 5.2: Slave requests synchronous EV3, EV4 OR EV5 connection. . 792
Figure 5.3: Master requests synchronous connection using SCO. ..... 792
Figure 5.4: Master requests synchronous connection with legacy slave. . 793
Figure 5.5: Any device that supports only SCO connections requests a synchronous connection with a device. ..... 793
Figure 5.6: Master renegotiates eSCO connection. ..... 794
Figure 5.7: Slave renegotiates eSCO connection. ..... 794
Figure 5.8: Synchronous disconnection of eSCO connection. ..... 795
Figure 5.9: Synchronous disconnection of SCO connection. ..... 795
Figure 6.1: Sniff mode request. ..... 796
Figure 6.2: Exit sniff mode request. ..... 796
Figure 6.3: Hold request. ..... 797
Figure 6.4: Master forces hold mode. ..... 797
Figure 6.5: Slave forces hold mode. ..... 798
Figure 6.6: Hold mode completes. ..... 798
Figure 6.7: Park state request from master. ..... 799
Figure 6.8: Park state request from slave. ..... 799
Figure 6.9: Master unparks slave for supervision. ..... 800
Figure 6.10: Master exits park state with slave. ..... 800
Figure 6.11: Slave exits park state with master. ..... 801
Figure 7.1: Host to Controller flow control. ..... 802
Figure 7.2: Controller to Host flow control. ..... 803
Figure 8.1: Entering local loopback mode. ..... 804
Figure 8.2: Looping back data in local loopback mode. ..... 804
Figure 8.3: Looping back commands in local loopback mode. ..... 805
Figure 8.4: Exiting local loopback mode. ..... 805
Figure 8.5: Entering remote loopback mode. ..... 806
Figure 8.6: Looping back data in Remote Loopback Mode. ..... 806
Figure 8.7: Exiting remote loopback mode. ..... 807

# SAMPLE DATA

***This appendix contains sample data for various parts of the Bluetooth baseband specification. All sample data are provided for reference purpose only; they are intended as a complement to the definitions provided elsewhere in the specification. They can be used to check the behavior of an implementation and avoid misunderstandings. Fulfilling these sample data is a necessary but not sufficient condition for an implementation to be fully Bluetooth compliant.***


Ford Ex. 1052

*Sample Data*

# CONTENTS

**1 Encryption Sample Data ..... 953**
1.1 Generating Kc' from Kc, ..... 953
1.2 First Set of Sample Data ..... 956
1.3 Second Set of Sample Data ..... 964
1.4 Third Set of Samples ..... 972
1.5 Fourth Set of Samples ..... 980
**2 Frequency Hopping Sample Data ..... 988**
2.1 First set ..... 989
2.2 Second set ..... 995
2.3 Third set ..... 1001
**3 Access Code Sample Data ..... 1007**
**4 HEC and Packet Header Sample Data ..... 1010**
**5 CRC Sample Data ..... 1011**
**6 Complete Sample Packets ..... 1012**
6.1 Example of DH1 Packet ..... 1012
6.2 Example of DM1 Packet ..... 1013
**7 Simple Pairing Sample Data ..... 1014**
7.1 P-192 Sample Data ..... 1014
7.1.1 Data Set 1 ..... 1014
7.1.2 Data Set 2 ..... 1014
7.1.3 Data Set 3 ..... 1014
7.1.4 Data Set 4 ..... 1014
7.1.5 Data Set 5 ..... 1014
7.1.6 Data Set 6 ..... 1015
7.1.7 Data Set 7 ..... 1015
7.1.8 Data Set 8 ..... 1015
7.1.9 Data Set 9 ..... 1015
7.1.10 Data Set 10 ..... 1015
7.2 Hash Functions Sample Data ..... 1015
7.2.1 f1() ..... 1016
7.2.2 g() ..... 1017
7.2.3 f2() ..... 1017
7.2.4 f3() ..... 1017
7.2.5 h2() ..... 1024
**8 Whitening Sequence Sample Data ..... 1026**
**9 FEC Sample Data ..... 1029**

**10 Encryption Key Sample Data ..... 1030**

10.1 Four Tests of E1 ..... 1030

10.2 Four Tests of E21 ..... 1035

10.3 Three Tests of E22 ..... 1037

10.4 Tests of E22 With Pin Augmenting ..... 1039

10.5 Four Tests of E3 ..... 1049

# 1 ENCRYPTION SAMPLE DATA

This section contains four sets of sample data for the encryption process.

With respect to the functional description of the encryption engine in the Bluetooth baseband specification, the contents of registers and resulting concurrent values are listed as well. This by no means excludes different implementations (as far as they produce the same encryption stream) but is intended to describe the functional behavior.

In case of misunderstandings or inconsistencies, these sample data form the normative reference.

## 1.1 GENERATING KC' FROM KC,

where Kc'(x) = g2(x)(Kc(x) mod g1(x)).

```
Note: All polynomials are in hexadecimal notation.
      'L' is the effective key length in bytes.
      The notation 'p: [m]' implies that deg(p(x)) = m.

                              MSB                                      LSB
L = 1
g1:                [8]        00000000 00000000 00000000 0000011d
g2:                [119]      00e275a0 abd218d4 cf928b9b bf6cb08f
Kc:                           a2b230a4 93f281bb 61a85b82 a9d4a30e
Kc mod g1:         [7]        00000000 00000000 00000000 0000009f
g2(Kc mod g1):     [126]      7aa16f39 59836ba3 22049a7b 87f1d8a5
-------------------------------------------------------------
L = 2
g1:                [16]       00000000 00000000 00000000 0001003f
g2:                [112]      0001e3f6 3d7659b3 7f18c258 cff6efef
Kc:                           64e7df78 bb7ccaa4 61433123 5b3222ad
Kc mod g1:         [12]       00000000 00000000 00000000 00001ff0
g2(Kc mod g1):     [124]      142057bb 0bceac4c 58bd142e 1e710a50
-------------------------------------------------------------
L = 3
g1:                [24]       00000000 00000000 00000000 010000db
g2:                [104]      000001be f66c6c3a b1030a5a 1919808b
Kc:                           575e5156 ba685dc6 112124ac edb2c179
Kc mod g1:         [23]       00000000 00000000 00000000 008ddbc8
g2(Kc mod g1):     [127]      d56d0adb 8216cb39 7fe3c591 1ff95618
-------------------------------------------------------------
L = 4
g1:                [32]       00000000 00000000 00000001 000000af
g2:                [96]       00000001 6ab89969 de17467f d3736ad9
Kc:                           8917b4fc 403b6db2 1596b86d 1cb8adab
Kc mod g1:         [31]       00000000 00000000 00000000 aa1e78aa
g2(Kc mod g1):     [127]      91910128 b0e2f5ed a132a03e af3d8cda
-------------------------------------------------------------
```

*Sample Data*

```
L = 5
g1:                [40]        00000000 00000000 00000100 00000039
g2:                [88]        00000000 01630632 91da50ec 55715247
Kc:                            785c915b dd25b9c6 0102ab00 b6cd2a68
Kc mod g1:         [38]        00000000 00000000 0000007f 13d44436
g2(Kc mod g1):     [126]       6fb5651c cb80c8d7 ea1ee56d f1ec5d02
------------------------------------------------------------
L = 6
g1:                [48]        00000000 00000000 00010000 00000291
g2:                [77]        00000000 00002c93 52aa6cc0 54468311
Kc:                            5e77d19f 55ccd7d5 798f9a32 3b83e5d8
Kc mod g1:         [47]        00000000 00000000 000082eb 4af213ed
g2(Kc mod g1):     [124]       16096bcb afcf8def 1d226a1b 4d3f9a3d
------------------------------------------------------------
```

*Sample Data*

```
L = 7
g1:                [56]        00000000 00000000 01000000 00000095
g2:                [71]        00000000 000000b3 f7fffce2 79f3a073
Kc:                            05454e03 8ddcfbe3 ed024b2d 92b7f54c
Kc mod g1:         [55]        00000000 00000000 0095b8a4 8eb816da
g2(Kc mod g1):     [126]       50f9c0d4 e3178da9 4a09fe0d 34f67b0e
------------------------------------------------------------
L = 8
g1:                [64]        00000000 00000001 00000000 0000001b
g2:                [63]        00000000 00000000 a1ab815b c7ec8025
Kc:                            7ce149fc f4b38ad7 2a5d8a41 eb15ba31
Kc mod g1:         [63]        00000000 00000000 8660806c 1865deec
g2(Kc mod g1):     [126]       532c36d4 5d0954e0 922989b6 826f78dc
------------------------------------------------------------
L = 9
g1:                [72]        00000000 00000100 00000000 00000609
g2:                [49]        00000000 00000000 0002c980 11d8b04d
Kc:                            5eeff7ca 84fc2782 9c051726 3df6f36e
Kc mod g1:         [71]        00000000 00000083 58ccb7d0 b95d3c71
g2(Kc mod g1):     [120]       016313f6 0d3771cf 7f8e4bb9 4aa6827d
------------------------------------------------------------
L = 10
g1:                [80]        00000000 00010000 00000000 00000215
g2:                [42]        00000000 00000000 0000058e 24f9a4bb
Kc:                            7b13846e 88beb4de 34e7160a fd44dc65
Kc mod g1:         [79]        00000000 0000b4de 34171767 f36981c3
g2(Kc mod g1):     [121]       023bc1ec 34a0029e f798dcfb 618ba58d
------------------------------------------------------------
L = 11
g1:                [88]        00000000 01000000 00000000 0000013b
g2:                [35]        00000000 00000000 0000000c a76024d7
Kc:                            bda6de6c 6e7d757e 8dfe2d49 9a181193
Kc mod g1:         [86]        00000000 007d757e 8dfe88aa 2fcee371
g2(Kc mod g1):     [121]       022e08a9 3aa51d8d 2f93fa78 85cc1f87
------------------------------------------------------------
L = 12
g1:                [96]        00000001 00000000 00000000 000000dd
g2:                [28]        00000000 00000000 00000000 1c9c26b9
Kc:                            e6483b1c 2cdb1040 9a658f97 c4efd90d
Kc mod g1:         [93]        00000000 2cdb1040 9a658fd7 5b562e41
g2(Kc mod g1):     [121]       030d752b 216fe29b b880275c d7e6f6f9
------------------------------------------------------------
L = 13
g1:                [104]       00000100 00000000 00000000 0000049d
g2:                [21]        00000000 00000000 00000000 0026d9e3
Kc:                            d79d281d a2266847 6b223c46 dc0ab9ee
Kc mod g1:         [100]       0000001d a2266847 6b223c45 e1fc5fa6
g2(Kc mod g1):     [121]       03f11138 9cebf919 00b93808 4ac158aa
------------------------------------------------------------
```

*Sample Data*

```
L = 14
g1:                [112]       00010000 00000000 00000000 0000014f
g2:                [14]        00000000 00000000 00000000 00004377
Kc:                            cad9a65b 9fca1c1d a2320fcf 7c4ae48e
Kc mod g1:         [111]       0000a65b 9fca1c1d a2320fcf 7cb6a909
g2(Kc mod g1):     [125]       284840fd f1305f3c 529f5703 76adf7cf
------------------------------------------------------------
L = 15
g1:                [120]       01000000 00000000 00000000 000000e7
g2:                [7]         00000000 00000000 00000000 00000089
Kc:                            21f0cc31 049b7163 d375e9e1 06029809
Kc mod g1:         [119]       00f0cc31 049b7163 d375e9e1 0602840e
g2(Kc mod g1):     [126]       7f10b53b 6df84b94 f22e566a 3754a37e
------------------------------------------------------------
L = 16
g1:                [128]       00000001 00000000 00000000 00000000 00000000
g2:                [0]         00000000 00000000 00000000 00000001
Kc:                            35ec8fc3 d50ccd32 5f2fd907 bde206de
Kc mod g1:         [125]       35ec8fc3 d50ccd32 5f2fd907 bde206de
g2(Kc mod g1):     [125]       35ec8fc3 d50ccd32 5f2fd907 bde206de
------------------------------------------------------------
```

## 1.2 FIRST SET OF SAMPLE DATA

```
Initial values for the key, pan address and clock

K'c1[0]   = 00   K'c1[1]   = 00    K'c1[2]   = 00    K'c1[3]  = 00
K'c1[4]   = 00   K'c1[5]   = 00    K'c1[6]   = 00    K'c1[7]  = 00
K'c1[8]   = 00   K'c1[9]   = 00    K'c1[10]  = 00    K'c1[11] = 00
K'c1[12]  = 00   K'c1[13]  = 00    K'c1[14]  = 00    K'c1[15] = 00

Addr1[0] = 00   Addr1[1] = 00    Addr1[2] = 00
Addr1[3] = 00   Addr1[4] = 00    Addr1[5] = 00

CL1[0]  = 00   CL1[1]  = 00    CL1[2]  = 00    CL1[3]  = 00


=============================================================================================
         Fill LFSRs with initial data
=============================================================================================

  t  clk#    LFSR1    LFSR2     LFSR3      LFSR4           X1 X2 X3 X4   Z  C[t+1] C[t] C[t-1]

  0     0    0000000* 00000000* 000000000* 0000000000*     0  0  0  0    0  00     00   00
  1     1    0000000* 00000001* 000000000* 0000000001*     0  0  0  0    0  00     00   00
  2     2    0000000* 00000002* 000000000* 0000000003*     0  0  0  0    0  00     00   00
  3     3    0000000* 00000004* 000000000* 0000000007*     0  0  0  0    0  00     00   00
  4     4    0000000* 00000008* 000000000* 000000000E*     0  0  0  0    0  00     00   00
  5     5    0000000* 00000010* 000000000* 000000001C*     0  0  0  0    0  00     00   00
  6     6    0000000* 00000020* 000000000* 0000000038*     0  0  0  0    0  00     00   00
```

*Sample Data*

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 7 | 0000000* | 00000040* | 000000000* | 0000000070* | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 8 | 8 | 0000000* | 00000080* | 000000000* | 00000000E0* | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 9 | 9 | 0000000* | 00000100* | 000000000* | 00000001C0* | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 10 | 10 | 0000000* | 00000200* | 000000000* | 0000000380* | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 11 | 11 | 0000000* | 00000400* | 000000000* | 0000000700* | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 12 | 12 | 0000000* | 00000800* | 000000000* | 0000000E00* | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 13 | 13 | 0000000* | 00001000* | 000000000* | 0000001C00* | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 14 | 14 | 0000000* | 00002000* | 000000000* | 0000003800* | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 15 | 15 | 0000000* | 00004000* | 000000000* | 0000007000* | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 16 | 16 | 0000000* | 00008000* | 000000000* | 000000E000* | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 17 | 17 | 0000000* | 00010000* | 000000000* | 000001C000* | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 18 | 18 | 0000000* | 00020000* | 000000000* | 0000038000* | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 19 | 19 | 0000000* | 00040000* | 000000000* | 0000070000* | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 20 | 20 | 0000000* | 00080000* | 000000000* | 00000E0000* | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 21 | 21 | 0000000* | 00100000* | 000000000* | 00001C0000* | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 22 | 22 | 0000000* | 00200000* | 000000000* | 0000380000* | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 23 | 23 | 0000000* | 00400000* | 000000000* | 0000700000* | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 24 | 24 | 0000000* | 00800000* | 000000000* | 0000E00000* | 0 | 1 | 0 | 0 | 1 | 00 | 00 | 00 |
| 25 | 25 | 0000000* | 01000000* | 000000000* | 0001C00000* | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 26 | 26 | 0000000 | 02000000* | 000000000* | 0003800000* | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 27 | 27 | 0000000 | 04000000* | 000000000* | 0007000000* | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 28 | 28 | 0000000 | 08000000* | 000000000* | 000E000000* | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 29 | 29 | 0000000 | 10000000* | 000000000* | 001C000000* | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 30 | 30 | 0000000 | 20000000* | 000000000* | 0038000000* | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 31 | 31 | 0000000 | 40000000* | 000000000* | 0070000000* | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 32 | 32 | 0000000 | 00000001 | 000000000* | 00E0000000* | 0 | 0 | 0 | 1 | 1 | 00 | 00 | 00 |
| 33 | 33 | 0000000 | 00000002 | 000000000* | 01C0000000* | 0 | 0 | 0 | 1 | 1 | 00 | 00 | 00 |
| 34 | 34 | 0000000 | 00000004 | 000000000 | 0380000000* | 0 | 0 | 0 | 1 | 1 | 00 | 00 | 00 |
| 35 | 35 | 0000000 | 00000008 | 000000000 | 0700000000* | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 36 | 36 | 0000000 | 00000010 | 000000000 | 0E00000000* | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 37 | 37 | 0000000 | 00000020 | 000000000 | 1C00000000* | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 38 | 38 | 0000000 | 00000040 | 000000000 | 3800000000* | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 39 | 39 | 0000000 | 00000080 | 000000000 | 7000000000* | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |

Start clocking Summation Combiner

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 40 | 1 | 0000000 | 00000100 | 000000000 | 6000000001 | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 41 | 2 | 0000000 | 00000200 | 000000000 | 4000000003 | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 42 | 3 | 0000000 | 00000400 | 000000000 | 0000000007 | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 43 | 4 | 0000000 | 00000800 | 000000000 | 000000000E | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 44 | 5 | 0000000 | 00001001 | 000000000 | 000000001D | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 45 | 6 | 0000000 | 00002002 | 000000000 | 000000003B | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 46 | 7 | 0000000 | 00004004 | 000000000 | 0000000077 | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 47 | 8 | 0000000 | 00008008 | 000000000 | 00000000EE | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 48 | 9 | 0000000 | 00010011 | 000000000 | 00000001DD | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 49 | 10 | 0000000 | 00020022 | 000000000 | 00000003BB | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 50 | 11 | 0000000 | 00040044 | 000000000 | 0000000777 | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 51 | 12 | 0000000 | 00080088 | 000000000 | 0000000EEE | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 52 | 13 | 0000000 | 00100110 | 000000000 | 0000001DDD | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 53 | 14 | 0000000 | 00200220 | 000000000 | 0000003BBB | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 54 | 15 | 0000000 | 00400440 | 000000000 | 0000007777 | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 55 | 16 | 0000000 | 00800880 | 000000000 | 000000EEEE | 0 | 1 | 0 | 0 | 1 | 00 | 00 | 00 |
| 56 | 17 | 0000000 | 01001100 | 000000000 | 000001DDDD | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 57 | 18 | 0000000 | 02002200 | 000000000 | 000003BBBB | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 58 | 19 | 0000000 | 04004400 | 000000000 | 0000077777 | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 59 | 20 | 0000000 | 08008800 | 000000000 | 00000EEEEE | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 60 | 21 | 0000000 | 10011000 | 000000000 | 00001DDDDD | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |

*Sample Data*

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 61 | 22 | 0000000 | 20022000 | 000000000 | 00003BBBBB | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 62 | 23 | 0000000 | 40044000 | 000000000 | 0000777777 | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 63 | 24 | 0000000 | 00088001 | 000000000 | 0000EEEEEE | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 64 | 25 | 0000000 | 00110003 | 000000000 | 0001DDDDDD | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 65 | 26 | 0000000 | 00220006 | 000000000 | 0003BBBBBB | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 66 | 27 | 0000000 | 0044000C | 000000000 | 0007777777 | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 67 | 28 | 0000000 | 00880018 | 000000000 | 000EEEEEEE | 0 | 1 | 0 | 0 | 1 | 00 | 00 | 00 |
| 68 | 29 | 0000000 | 01100031 | 000000000 | 001DDDDDDC | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 69 | 30 | 0000000 | 02200062 | 000000000 | 003BBBBBB8 | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 70 | 31 | 0000000 | 044000C4 | 000000000 | 0077777770 | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 71 | 32 | 0000000 | 08800188 | 000000000 | 00EEEEEEE0 | 0 | 1 | 0 | 1 | 0 | 01 | 00 | 00 |
| 72 | 33 | 0000000 | 11000311 | 000000000 | 01DDDDDDC1 | 0 | 0 | 0 | 1 | 0 | 00 | 01 | 00 |
| 73 | 34 | 0000000 | 22000622 | 000000000 | 03BBBBBB83 | 0 | 0 | 0 | 1 | 1 | 11 | 00 | 01 |
| 74 | 35 | 0000000 | 44000C44 | 000000000 | 0777777707 | 0 | 0 | 0 | 0 | 1 | 10 | 11 | 00 |
| 75 | 36 | 0000000 | 08001888 | 000000000 | 0EEEEEEE0E | 0 | 0 | 0 | 1 | 1 | 01 | 10 | 11 |
| 76 | 37 | 0000000 | 10003111 | 000000000 | 1DDDDDDC1D | 0 | 0 | 0 | 1 | 0 | 01 | 01 | 10 |
| 77 | 38 | 0000000 | 20006222 | 000000000 | 3BBBBBB83B | 0 | 0 | 0 | 1 | 0 | 11 | 01 | 01 |
| 78 | 39 | 0000000 | 4000C444 | 000000000 | 7777777077 | 0 | 0 | 0 | 0 | 1 | 01 | 11 | 01 |
| 79 | 40 | 0000000 | 00018888 | 000000000 | 6EEEEEE0EF | 0 | 0 | 0 | 1 | 0 | 10 | 01 | 11 |
| 80 | 41 | 0000000 | 00031110 | 000000000 | 5DDDDDC1DE | 0 | 0 | 0 | 1 | 1 | 00 | 10 | 01 |
| 81 | 42 | 0000000 | 00062220 | 000000000 | 3BBBBB83BC | 0 | 0 | 0 | 1 | 1 | 01 | 00 | 10 |
| 82 | 43 | 0000000 | 000C4440 | 000000000 | 7777770779 | 0 | 0 | 0 | 0 | 1 | 01 | 01 | 00 |
| 83 | 44 | 0000000 | 00188880 | 000000000 | 6EEEEE0EF2 | 0 | 0 | 0 | 1 | 0 | 11 | 01 | 01 |
| 84 | 45 | 0000000 | 00311100 | 000000000 | 5DDDDC1DE5 | 0 | 0 | 0 | 1 | 0 | 10 | 11 | 01 |
| 85 | 46 | 0000000 | 00622200 | 000000000 | 3BBBB83BCB | 0 | 0 | 0 | 1 | 1 | 01 | 10 | 11 |
| 86 | 47 | 0000000 | 00C44400 | 000000000 | 7777707797 | 0 | 1 | 0 | 0 | 0 | 01 | 01 | 10 |
| 87 | 48 | 0000000 | 01888801 | 000000000 | 6EEEE0EF2F | 0 | 1 | 0 | 1 | 1 | 11 | 01 | 01 |
| 88 | 49 | 0000000 | 03111003 | 000000000 | 5DDDC1DE5E | 0 | 0 | 0 | 1 | 0 | 10 | 11 | 01 |
| 89 | 50 | 0000000 | 06222006 | 000000000 | 3BBB83BCBC | 0 | 0 | 0 | 1 | 1 | 01 | 10 | 11 |
| 90 | 51 | 0000000 | 0C44400C | 000000000 | 7777077979 | 0 | 0 | 0 | 0 | 1 | 00 | 01 | 10 |
| 91 | 52 | 0000000 | 18888018 | 000000000 | 6EEE0EF2F2 | 0 | 1 | 0 | 1 | 0 | 10 | 00 | 01 |
| 92 | 53 | 0000000 | 31110030 | 000000000 | 5DDC1DE5E5 | 0 | 0 | 0 | 1 | 1 | 11 | 10 | 00 |
| 93 | 54 | 0000000 | 62220060 | 000000000 | 3BB83BCBCB | 0 | 0 | 0 | 1 | 0 | 00 | 11 | 10 |
| 94 | 55 | 0000000 | 444400C1 | 000000000 | 7770779797 | 0 | 0 | 0 | 0 | 0 | 10 | 00 | 11 |
| 95 | 56 | 0000000 | 08880183 | 000000000 | 6EE0EF2F2F | 0 | 1 | 0 | 1 | 0 | 00 | 10 | 00 |
| 96 | 57 | 0000000 | 11100307 | 000000000 | 5DC1DE5E5F | 0 | 0 | 0 | 1 | 1 | 01 | 00 | 10 |
| 97 | 58 | 0000000 | 2220060E | 000000000 | 3B83BCBCBF | 0 | 0 | 0 | 1 | 0 | 00 | 01 | 00 |
| 98 | 59 | 0000000 | 44400C1C | 000000000 | 770779797E | 0 | 0 | 0 | 0 | 0 | 11 | 00 | 01 |
| 99 | 60 | 0000000 | 08801838 | 000000000 | 6E0EF2F2FC | 0 | 1 | 0 | 0 | 0 | 01 | 11 | 00 |
| 100 | 61 | 0000000 | 11003070 | 000000000 | 5C1DE5E5F8 | 0 | 0 | 0 | 0 | 1 | 11 | 01 | 11 |
| 101 | 62 | 0000000 | 220060E0 | 000000000 | 383BCBCBF0 | 0 | 0 | 0 | 0 | 1 | 01 | 11 | 01 |
| 102 | 63 | 0000000 | 4400C1C0 | 000000000 | 70779797E0 | 0 | 0 | 0 | 0 | 1 | 11 | 01 | 11 |
| 103 | 64 | 0000000 | 08018380 | 000000000 | 60EF2F2FC1 | 0 | 0 | 0 | 1 | 0 | 10 | 11 | 01 |
| 104 | 65 | 0000000 | 10030701 | 000000000 | 41DE5E5F82 | 0 | 0 | 0 | 1 | 1 | 01 | 10 | 11 |
| 105 | 66 | 0000000 | 20060E02 | 000000000 | 03BCBCBF04 | 0 | 0 | 0 | 1 | 0 | 01 | 01 | 10 |
| 106 | 67 | 0000000 | 400C1C05 | 000000000 | 0779797E09 | 0 | 0 | 0 | 0 | 1 | 10 | 01 | 01 |
| 107 | 68 | 0000000 | 0018380A | 000000000 | 0EF2F2FC12 | 0 | 0 | 0 | 1 | 1 | 00 | 10 | 01 |
| 108 | 69 | 0000000 | 00307015 | 000000000 | 1DE5E5F825 | 0 | 0 | 0 | 1 | 1 | 01 | 00 | 10 |
| 109 | 70 | 0000000 | 0060E02A | 000000000 | 3BCBCBF04B | 0 | 0 | 0 | 1 | 0 | 00 | 01 | 00 |
| 110 | 71 | 0000000 | 00C1C055 | 000000000 | 779797E097 | 0 | 1 | 0 | 1 | 0 | 10 | 00 | 01 |
| 111 | 72 | 0000000 | 018380AA | 000000000 | 6F2F2FC12F | 0 | 1 | 0 | 0 | 1 | 11 | 10 | 00 |
| 112 | 73 | 0000000 | 03070154 | 000000000 | 5E5E5F825E | 0 | 0 | 0 | 0 | 1 | 11 | 11 | 10 |
| 113 | 74 | 0000000 | 060E02A8 | 000000000 | 3CBCBF04BC | 0 | 0 | 0 | 1 | 0 | 11 | 11 | 11 |
| 114 | 75 | 0000000 | 0C1C0550 | 000000000 | 79797E0979 | 0 | 0 | 0 | 0 | 1 | 00 | 11 | 11 |
| 115 | 76 | 0000000 | 18380AA0 | 000000000 | 72F2FC12F2 | 0 | 0 | 0 | 1 | 1 | 10 | 00 | 11 |
| 116 | 77 | 0000000 | 30701541 | 000000000 | 65E5F825E5 | 0 | 0 | 0 | 1 | 1 | 11 | 10 | 00 |
| 117 | 78 | 0000000 | 60E02A82 | 000000000 | 4BCBF04BCB | 0 | 1 | 0 | 1 | 1 | 00 | 11 | 10 |

*Sample Data*

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 118 | 79 | 0000000 | 41C05505 | 000000000 | 1797E09796 | 0 | 1 | 0 | 1 | 0 | 11 | 00 | 11 |
| 119 | 80 | 0000000 | 0380AA0A | 000000000 | 2F2FC12F2C | 0 | 1 | 0 | 0 | 0 | 01 | 11 | 00 |
| 120 | 81 | 0000000 | 07015415 | 000000000 | 5E5F825E59 | 0 | 0 | 0 | 0 | 1 | 11 | 01 | 11 |
| 121 | 82 | 0000000 | 0E02A82A | 000000000 | 3CBF04BCB2 | 0 | 0 | 0 | 1 | 0 | 10 | 11 | 01 |
| 122 | 83 | 0000000 | 1C055054 | 000000000 | 797E097964 | 0 | 0 | 0 | 0 | 0 | 01 | 10 | 11 |
| 123 | 84 | 0000000 | 380AA0A8 | 000000000 | 72FC12F2C9 | 0 | 0 | 0 | 1 | 0 | 01 | 01 | 10 |
| 124 | 85 | 0000000 | 70154151 | 000000000 | 65F825E593 | 0 | 0 | 0 | 1 | 0 | 11 | 01 | 01 |
| 125 | 86 | 0000000 | 602A82A3 | 000000000 | 4BF04BCB26 | 0 | 0 | 0 | 1 | 0 | 10 | 11 | 01 |
| 126 | 87 | 0000000 | 40550546 | 000000000 | 17E097964C | 0 | 0 | 0 | 1 | 1 | 01 | 10 | 11 |
| 127 | 88 | 0000000 | 00AA0A8D | 000000000 | 2FC12F2C99 | 0 | 1 | 0 | 1 | 1 | 01 | 01 | 10 |
| 128 | 89 | 0000000 | 0154151A | 000000000 | 5F825E5932 | 0 | 0 | 0 | 1 | 0 | 11 | 01 | 01 |
| 129 | 90 | 0000000 | 02A82A34 | 000000000 | 3F04BCB264 | 0 | 1 | 0 | 0 | 0 | 10 | 11 | 01 |
| 130 | 91 | 0000000 | 05505468 | 000000000 | 7E097964C9 | 0 | 0 | 0 | 0 | 0 | 01 | 10 | 11 |
| 131 | 92 | 0000000 | 0AA0A8D0 | 000000000 | 7C12F2C992 | 0 | 1 | 0 | 0 | 0 | 01 | 01 | 10 |
| 132 | 93 | 0000000 | 154151A1 | 000000000 | 7825E59324 | 0 | 0 | 0 | 0 | 1 | 10 | 01 | 01 |
| 133 | 94 | 0000000 | 2A82A342 | 000000000 | 704BCB2648 | 0 | 1 | 0 | 0 | 1 | 00 | 10 | 01 |
| 134 | 95 | 0000000 | 55054684 | 000000000 | 6097964C91 | 0 | 0 | 0 | 1 | 1 | 01 | 00 | 10 |
| 135 | 96 | 0000000 | 2A0A8D09 | 000000000 | 412F2C9923 | 0 | 0 | 0 | 0 | 1 | 01 | 01 | 00 |
| 136 | 97 | 0000000 | 54151A12 | 000000000 | 025E593246 | 0 | 0 | 0 | 0 | 1 | 10 | 01 | 01 |
| 137 | 98 | 0000000 | 282A3424 | 000000000 | 04BCB2648D | 0 | 0 | 0 | 1 | 1 | 00 | 10 | 01 |
| 138 | 99 | 0000000 | 50546848 | 000000000 | 097964C91A | 0 | 0 | 0 | 0 | 0 | 01 | 00 | 10 |
| 139 | 100 | 0000000 | 20A8D090 | 000000000 | 12F2C99235 | 0 | 1 | 0 | 1 | 1 | 00 | 01 | 00 |
| 140 | 101 | 0000000 | 4151A120 | 000000000 | 25E593246A | 0 | 0 | 0 | 1 | 1 | 11 | 00 | 01 |
| 141 | 102 | 0000000 | 02A34240 | 000000000 | 4BCB2648D5 | 0 | 1 | 0 | 1 | 1 | 01 | 11 | 00 |
| 142 | 103 | 0000000 | 05468481 | 000000000 | 17964C91AB | 0 | 0 | 0 | 1 | 0 | 10 | 01 | 11 |
| 143 | 104 | 0000000 | 0A8D0903 | 000000000 | 2F2C992357 | 0 | 1 | 0 | 0 | 1 | 00 | 10 | 01 |
| 144 | 105 | 0000000 | 151A1206 | 000000000 | 5E593246AE | 0 | 0 | 0 | 0 | 0 | 01 | 00 | 10 |
| 145 | 106 | 0000000 | 2A34240C | 000000000 | 3CB2648D5C | 0 | 0 | 0 | 1 | 0 | 00 | 01 | 00 |
| 146 | 107 | 0000000 | 54684818 | 000000000 | 7964C91AB8 | 0 | 0 | 0 | 0 | 0 | 11 | 00 | 01 |
| 147 | 108 | 0000000 | 28D09030 | 000000000 | 72C9923571 | 0 | 1 | 0 | 1 | 1 | 01 | 11 | 00 |
| 148 | 109 | 0000000 | 51A12060 | 000000000 | 6593246AE2 | 0 | 1 | 0 | 1 | 1 | 10 | 01 | 11 |
| 149 | 110 | 0000000 | 234240C0 | 000000000 | 4B2648D5C5 | 0 | 0 | 0 | 0 | 0 | 00 | 10 | 01 |
| 150 | 111 | 0000000 | 46848180 | 000000000 | 164C91AB8A | 0 | 1 | 0 | 0 | 1 | 01 | 00 | 10 |
| 151 | 112 | 0000000 | 0D090301 | 000000000 | 2C99235714 | 0 | 0 | 0 | 1 | 0 | 00 | 01 | 00 |
| 152 | 113 | 0000000 | 1A120602 | 000000000 | 593246AE28 | 0 | 0 | 0 | 0 | 0 | 11 | 00 | 01 |
| 153 | 114 | 0000000 | 34240C04 | 000000000 | 32648D5C51 | 0 | 0 | 0 | 0 | 1 | 10 | 11 | 00 |
| 154 | 115 | 0000000 | 68481809 | 000000000 | 64C91AB8A2 | 0 | 0 | 0 | 1 | 1 | 01 | 10 | 11 |
| 155 | 116 | 0000000 | 50903012 | 000000000 | 4992357144 | 0 | 1 | 0 | 1 | 1 | 01 | 01 | 10 |
| 156 | 117 | 0000000 | 21206024 | 000000000 | 13246AE288 | 0 | 0 | 0 | 0 | 1 | 10 | 01 | 01 |
| 157 | 118 | 0000000 | 4240C048 | 000000000 | 2648D5C511 | 0 | 0 | 0 | 0 | 0 | 00 | 10 | 01 |
| 158 | 119 | 0000000 | 04818090 | 000000000 | 4C91AB8A23 | 0 | 1 | 0 | 1 | 0 | 00 | 00 | 10 |
| 159 | 120 | 0000000 | 09030120 | 000000000 | 1923571446 | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 160 | 121 | 0000000 | 12060240 | 000000000 | 3246AE288D | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 161 | 122 | 0000000 | 240C0480 | 000000000 | 648D5C511B | 0 | 0 | 0 | 1 | 1 | 00 | 00 | 00 |
| 162 | 123 | 0000000 | 48180900 | 000000000 | 491AB8A237 | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 163 | 124 | 0000000 | 10301200 | 000000000 | 123571446F | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 164 | 125 | 0000000 | 20602400 | 000000000 | 246AE288DF | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 165 | 126 | 0000000 | 40C04800 | 000000000 | 48D5C511BE | 0 | 1 | 0 | 1 | 0 | 01 | 00 | 00 |
| 166 | 127 | 0000000 | 01809001 | 000000000 | 11AB8A237D | 0 | 1 | 0 | 1 | 1 | 00 | 01 | 00 |
| 167 | 128 | 0000000 | 03012002 | 000000000 | 23571446FA | 0 | 0 | 0 | 0 | 0 | 11 | 00 | 01 |
| 168 | 129 | 0000000 | 06024004 | 000000000 | 46AE288DF5 | 0 | 0 | 0 | 1 | 0 | 01 | 11 | 00 |
| 169 | 130 | 0000000 | 0C048008 | 000000000 | 0D5C511BEA | 0 | 0 | 0 | 0 | 1 | 11 | 01 | 11 |
| 170 | 131 | 0000000 | 18090011 | 000000000 | 1AB8A237D5 | 0 | 0 | 0 | 1 | 0 | 10 | 11 | 01 |
| 171 | 132 | 0000000 | 30120022 | 000000000 | 3571446FAA | 0 | 0 | 0 | 0 | 0 | 01 | 10 | 11 |
| 172 | 133 | 0000000 | 60240044 | 000000000 | 6AE288DF55 | 0 | 0 | 0 | 1 | 0 | 01 | 01 | 10 |
| 173 | 134 | 0000000 | 40480089 | 000000000 | 55C511BEAA | 0 | 0 | 0 | 1 | 0 | 11 | 01 | 01 |
| 174 | 135 | 0000000 | 00900113 | 000000000 | 2B8A237D54 | 0 | 1 | 0 | 1 | 1 | 10 | 11 | 01 |

*Sample Data*

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 175 | 136 | 0000000 | 01200227 | 000000000 | 571446FAA8 | 0 | 0 | 0 | 0 | 0 | 01 | 10 | 11 |
| 176 | 137 | 0000000 | 0240044E | 000000000 | 2E288DF550 | 0 | 0 | 0 | 0 | 1 | 00 | 01 | 10 |
| 177 | 138 | 0000000 | 0480089C | 000000000 | 5C511BEAA0 | 0 | 1 | 0 | 0 | 1 | 11 | 00 | 01 |
| 178 | 139 | 0000000 | 09001138 | 000000000 | 38A237D540 | 0 | 0 | 0 | 1 | 0 | 01 | 11 | 00 |
| 179 | 140 | 0000000 | 12002270 | 000000000 | 71446FAA81 | 0 | 0 | 0 | 0 | 1 | 11 | 01 | 11 |
| 180 | 141 | 0000000 | 240044E0 | 000000000 | 6288DF5503 | 0 | 0 | 0 | 1 | 0 | 10 | 11 | 01 |
| 181 | 142 | 0000000 | 480089C0 | 000000000 | 4511BEAA06 | 0 | 0 | 0 | 0 | 0 | 01 | 10 | 11 |
| 182 | 143 | 0000000 | 10011381 | 000000000 | 0A237D540D | 0 | 0 | 0 | 0 | 1 | 00 | 01 | 10 |
| 183 | 144 | 0000000 | 20022702 | 000000000 | 1446FAA81A | 0 | 0 | 0 | 0 | 0 | 11 | 00 | 01 |
| 184 | 145 | 0000000 | 40044E04 | 000000000 | 288DF55035 | 0 | 0 | 0 | 1 | 0 | 01 | 11 | 00 |
| 185 | 146 | 0000000 | 00089C08 | 000000000 | 511BEAA06A | 0 | 0 | 0 | 0 | 1 | 11 | 01 | 11 |
| 186 | 147 | 0000000 | 00113810 | 000000000 | 2237D540D5 | 0 | 0 | 0 | 0 | 1 | 01 | 11 | 01 |
| 187 | 148 | 0000000 | 00227021 | 000000000 | 446FAA81AA | 0 | 0 | 0 | 0 | 1 | 11 | 01 | 11 |
| 188 | 149 | 0000000 | 0044E042 | 000000000 | 08DF550355 | 0 | 0 | 0 | 1 | 0 | 10 | 11 | 01 |
| 189 | 150 | 0000000 | 0089C085 | 000000000 | 11BEAA06AA | 0 | 1 | 0 | 1 | 0 | 10 | 10 | 11 |
| 190 | 151 | 0000000 | 0113810A | 000000000 | 237D540D54 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 |
| 191 | 152 | 0000000 | 02270215 | 000000000 | 46FAA81AA9 | 0 | 0 | 0 | 1 | 1 | 10 | 10 | 10 |
| 192 | 153 | 0000000 | 044E042A | 000000000 | 0DF5503553 | 0 | 0 | 0 | 1 | 1 | 10 | 10 | 10 |
| 193 | 154 | 0000000 | 089C0854 | 000000000 | 1BEAA06AA7 | 0 | 1 | 0 | 1 | 0 | 01 | 10 | 10 |
| 194 | 155 | 0000000 | 113810A8 | 000000000 | 37D540D54E | 0 | 0 | 0 | 1 | 0 | 01 | 01 | 10 |
| 195 | 156 | 0000000 | 22702150 | 000000000 | 6FAA81AA9D | 0 | 0 | 0 | 1 | 0 | 11 | 01 | 01 |
| 196 | 157 | 0000000 | 44E042A0 | 000000000 | 5F5503553A | 0 | 1 | 0 | 0 | 0 | 10 | 11 | 01 |
| 197 | 158 | 0000000 | 09C08540 | 000000000 | 3EAA06AA75 | 0 | 1 | 0 | 1 | 0 | 10 | 10 | 11 |
| 198 | 159 | 0000000 | 13810A80 | 000000000 | 7D540D54EA | 0 | 1 | 0 | 0 | 1 | 10 | 10 | 10 |
| 199 | 160 | 0000000 | 27021500 | 000000000 | 7AA81AA9D5 | 0 | 0 | 0 | 1 | 1 | 10 | 10 | 10 |
| 200 | 161 | 0000000 | 4E042A00 | 000000000 | 75503553AB | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 |
| 201 | 162 | 0000000 | 1C085400 | 000000000 | 6AA06AA756 | 0 | 0 | 0 | 1 | 1 | 10 | 10 | 10 |
| 202 | 163 | 0000000 | 3810A800 | 000000000 | 5540D54EAC | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 |
| 203 | 164 | 0000000 | 70215000 | 000000000 | 2A81AA9D58 | 0 | 0 | 0 | 1 | 1 | 10 | 10 | 10 |
| 204 | 165 | 0000000 | 6042A001 | 000000000 | 5503553AB0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 |
| 205 | 166 | 0000000 | 40854002 | 000000000 | 2A06AA7561 | 0 | 1 | 0 | 0 | 1 | 10 | 10 | 10 |
| 206 | 167 | 0000000 | 010A8004 | 000000000 | 540D54EAC3 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 |
| 207 | 168 | 0000000 | 02150009 | 000000000 | 281AA9D586 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 |
| 208 | 169 | 0000000 | 042A0012 | 000000000 | 503553AB0C | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 |
| 209 | 170 | 0000000 | 08540024 | 000000000 | 206AA75618 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 |
| 210 | 171 | 0000000 | 10A80048 | 000000000 | 40D54EAC30 | 0 | 1 | 0 | 1 | 0 | 01 | 10 | 10 |
| 211 | 172 | 0000000 | 21500091 | 000000000 | 01AA9D5861 | 0 | 0 | 0 | 1 | 0 | 01 | 01 | 10 |
| 212 | 173 | 0000000 | 42A00122 | 000000000 | 03553AB0C3 | 0 | 1 | 0 | 0 | 0 | 11 | 01 | 01 |
| 213 | 174 | 0000000 | 05400244 | 000000000 | 06AA756186 | 0 | 0 | 0 | 1 | 0 | 10 | 11 | 01 |
| 214 | 175 | 0000000 | 0A800488 | 000000000 | 0D54EAC30D | 0 | 1 | 0 | 0 | 1 | 01 | 10 | 11 |
| 215 | 176 | 0000000 | 15000911 | 000000000 | 1AA9D5861A | 0 | 0 | 0 | 1 | 0 | 01 | 01 | 10 |
| 216 | 177 | 0000000 | 2A001223 | 000000000 | 3553AB0C35 | 0 | 0 | 0 | 0 | 1 | 10 | 01 | 01 |
| 217 | 178 | 0000000 | 54002446 | 000000000 | 6AA756186A | 0 | 0 | 0 | 1 | 1 | 00 | 10 | 01 |
| 218 | 179 | 0000000 | 2800488D | 000000000 | 554EAC30D5 | 0 | 0 | 0 | 0 | 0 | 01 | 00 | 10 |
| 219 | 180 | 0000000 | 5000911B | 000000000 | 2A9D5861AA | 0 | 0 | 0 | 1 | 0 | 00 | 01 | 00 |
| 220 | 181 | 0000000 | 20012236 | 000000000 | 553AB0C355 | 0 | 0 | 0 | 0 | 0 | 11 | 00 | 01 |
| 221 | 182 | 0000000 | 4002446C | 000000000 | 2A756186AA | 0 | 0 | 0 | 0 | 1 | 10 | 11 | 00 |
| 222 | 183 | 0000000 | 000488D9 | 000000000 | 54EAC30D54 | 0 | 0 | 0 | 1 | 1 | 01 | 10 | 11 |
| 223 | 184 | 0000000 | 000911B2 | 000000000 | 29D5861AA8 | 0 | 0 | 0 | 1 | 0 | 01 | 01 | 10 |
| 224 | 185 | 0000000 | 00122364 | 000000000 | 53AB0C3550 | 0 | 0 | 0 | 1 | 0 | 11 | 01 | 01 |
| 225 | 186 | 0000000 | 002446C8 | 000000000 | 2756186AA0 | 0 | 0 | 0 | 0 | 1 | 01 | 11 | 01 |
| 226 | 187 | 0000000 | 00488D90 | 000000000 | 4EAC30D540 | 0 | 0 | 0 | 1 | 0 | 10 | 01 | 11 |
| 227 | 188 | 0000000 | 00911B20 | 000000000 | 1D5861AA81 | 0 | 1 | 0 | 0 | 1 | 00 | 10 | 01 |
| 228 | 189 | 0000000 | 01223640 | 000000000 | 3AB0C35502 | 0 | 0 | 0 | 1 | 1 | 01 | 00 | 10 |
| 229 | 190 | 0000000 | 02446C80 | 000000000 | 756186AA05 | 0 | 0 | 0 | 0 | 1 | 01 | 01 | 00 |
| 230 | 191 | 0000000 | 0488D901 | 000000000 | 6AC30D540B | 0 | 1 | 0 | 1 | 1 | 11 | 01 | 01 |
| 231 | 192 | 0000000 | 0911B203 | 000000000 | 55861AA817 | 0 | 0 | 0 | 1 | 0 | 10 | 11 | 01 |

*Sample Data*

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 232 | 193 | 0000000 | 12236407 | 000000000 | 2B0C35502F | 0 | 0 | 0 | 0 | 0 | 01 | 10 | 11 |
| 233 | 194 | 0000000 | 2446C80E | 000000000 | 56186AA05F | 0 | 0 | 0 | 0 | 1 | 00 | 01 | 10 |
| 234 | 195 | 0000000 | 488D901C | 000000000 | 2C30D540BF | 0 | 1 | 0 | 0 | 1 | 11 | 00 | 01 |
| 235 | 196 | 0000000 | 111B2039 | 000000000 | 5861AA817E | 0 | 0 | 0 | 0 | 1 | 10 | 11 | 00 |
| 236 | 197 | 0000000 | 22364072 | 000000000 | 30C35502FD | 0 | 0 | 0 | 1 | 1 | 01 | 10 | 11 |
| 237 | 198 | 0000000 | 446C80E4 | 000000000 | 6186AA05FB | 0 | 0 | 0 | 1 | 0 | 01 | 01 | 10 |
| 238 | 199 | 0000000 | 08D901C8 | 000000000 | 430D540BF6 | 0 | 1 | 0 | 0 | 0 | 11 | 01 | 01 |
| 239 | 200 | 0000000 | 11B20391 | 000000000 | 061AA817EC | 0 | 1 | 0 | 0 | 0 | 10 | 11 | 01 |

```
Z[0]  = 3D
Z[1]  = C1
Z[2]  = F0
Z[3]  = BB
Z[4]  = 58
Z[5]  = 1E
Z[6]  = 42
Z[7]  = 42
Z[8]  = 4B
Z[9]  = 8E
Z[10] = C1
Z[11] = 2A
Z[12] = 40
Z[13] = 63
Z[14] = 7A
Z[15] = 1E

================================================================================================
         Reload this pattern into the LFSRs
         Hold content of Summation Combiner regs and calculate new C[t+1] and Z values
================================================================================================
LFSR1  <= 04B583D
LFSR2  <= 208E1EC1
LFSR3  <= 063C142F0
LFSR4  <= 0F7A2A42BB
C[t+1] <= 10

================================================================================================
         Generating 125 key symbols (encryption/decryption sequence)
================================================================================================
```

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 240 | 1 | 04B583D | 208E1EC1 | 063C142F0 | 0F7A2A42BB | 0 | 1 | 0 | 0 | 0 | 10 | 11 | 01 |
| 241 | 2 | 096B07A | 411C3D82 | 0C78285E1 | 1EF4548577 | 1 | 0 | 1 | 1 | 1 | 10 | 10 | 11 |
| 242 | 3 | 12D60F4 | 02387B04 | 18F050BC3 | 3DE8A90AEF | 0 | 0 | 1 | 1 | 0 | 01 | 10 | 10 |
| 243 | 4 | 05AC1E9 | 0470F609 | 11E0A1786 | 7BD15215DF | 0 | 0 | 0 | 1 | 0 | 01 | 01 | 10 |
| 244 | 5 | 0B583D2 | 08E1EC13 | 03C142F0C | 77A2A42BBF | 1 | 1 | 0 | 1 | 0 | 00 | 01 | 01 |
| 245 | 6 | 16B07A5 | 11C3D827 | 078285E18 | 6F4548577E | 0 | 1 | 0 | 0 | 1 | 11 | 00 | 01 |
| 246 | 7 | 0D60F4B | 2387B04F | 0F050BC30 | 5E8A90AEFD | 1 | 1 | 1 | 1 | 1 | 00 | 11 | 00 |
| 247 | 8 | 1AC1E97 | 470F609E | 1E0A17860 | 3D15215DFA | 1 | 0 | 1 | 0 | 0 | 11 | 00 | 11 |
| 248 | 9 | 1583D2E | 0E1EC13D | 1C142F0C0 | 7A2A42BBF4 | 0 | 0 | 1 | 0 | 0 | 01 | 11 | 00 |
| 249 | 10 | 0B07A5D | 1C3D827B | 18285E181 | 74548577E9 | 1 | 0 | 1 | 0 | 1 | 10 | 01 | 11 |
| 250 | 11 | 160F4BB | 387B04F7 | 1050BC302 | 68A90AEFD2 | 0 | 0 | 0 | 1 | 1 | 00 | 10 | 01 |
| 251 | 12 | 0C1E976 | 70F609EE | 00A178605 | 515215DFA5 | 1 | 1 | 0 | 0 | 0 | 00 | 00 | 10 |
| 252 | 13 | 183D2ED | 61EC13DD | 0142F0C0B | 22A42BBF4B | 1 | 1 | 0 | 1 | 1 | 01 | 00 | 00 |
| 253 | 14 | 107A5DA | 43D827BA | 0285E1817 | 4548577E97 | 0 | 1 | 0 | 0 | 0 | 00 | 01 | 00 |
| 254 | 15 | 00F4BB4 | 07B04F74 | 050BC302F | 0A90AEFD2E | 0 | 1 | 0 | 1 | 0 | 10 | 00 | 01 |
| 255 | 16 | 01E9769 | 0F609EE8 | 0A178605E | 15215DFA5C | 0 | 0 | 1 | 0 | 1 | 11 | 10 | 00 |
| 256 | 17 | 03D2ED3 | 1EC13DD0 | 142F0C0BD | 2A42BBF4B9 | 0 | 1 | 0 | 0 | 0 | 00 | 11 | 10 |
| 257 | 18 | 07A5DA7 | 3D827BA0 | 085E1817B | 548577E972 | 0 | 1 | 1 | 1 | 1 | 11 | 00 | 11 |

*Sample Data*

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 258 | 19 | 0F4BB4F | 7B04F740 | 10BC302F6 | 290AEFD2E5 | 1 | 0 | 0 | 0 | 0 | 01 | 11 | 00 |
| 259 | 20 | 1E9769F | 7609EE80 | 0178605ED | 5215DFA5CA | 1 | 0 | 0 | 0 | 0 | 10 | 01 | 11 |
| 260 | 21 | 1D2ED3F | 6C13DD01 | 02F0C0BDA | 242BBF4B94 | 1 | 0 | 0 | 0 | 1 | 00 | 10 | 01 |
| 261 | 22 | 1A5DA7E | 5827BA03 | 05E1817B4 | 48577E9729 | 1 | 0 | 0 | 0 | 1 | 01 | 00 | 10 |
| 262 | 23 | 14BB4FC | 304F7407 | 0BC302F69 | 10AEFD2E53 | 0 | 0 | 1 | 1 | 1 | 00 | 01 | 00 |
| 263 | 24 | 09769F9 | 609EE80E | 178605ED2 | 215DFA5CA7 | 1 | 1 | 0 | 0 | 0 | 10 | 00 | 01 |
| 264 | 25 | 12ED3F2 | 413DD01C | 0F0C0BDA4 | 42BBF4B94F | 0 | 0 | 1 | 1 | 0 | 00 | 10 | 00 |
| 265 | 26 | 05DA7E5 | 027BA038 | 1E1817B49 | 0577E9729F | 0 | 0 | 1 | 0 | 1 | 01 | 00 | 10 |
| 266 | 27 | 0BB4FCA | 04F74071 | 1C302F693 | 0AEFD2E53F | 1 | 1 | 1 | 1 | 1 | 11 | 01 | 00 |
| 267 | 28 | 1769F95 | 09EE80E3 | 18605ED27 | 15DFA5CA7F | 0 | 1 | 1 | 1 | 0 | 11 | 11 | 01 |
| 268 | 29 | 0ED3F2B | 13DD01C6 | 10C0BDA4F | 2BBF4B94FE | 1 | 1 | 0 | 1 | 0 | 10 | 11 | 11 |
| 269 | 30 | 1DA7E56 | 27BA038D | 01817B49F | 577E9729FD | 1 | 1 | 0 | 0 | 0 | 10 | 10 | 11 |
| 270 | 31 | 1B4FCAD | 4F74071B | 0302F693E | 2EFD2E53FB | 1 | 0 | 0 | 1 | 0 | 01 | 10 | 10 |
| 271 | 32 | 169F95B | 1EE80E37 | 0605ED27D | 5DFA5CA7F7 | 0 | 1 | 0 | 1 | 1 | 01 | 01 | 10 |
| 272 | 33 | 0D3F2B7 | 3DD01C6E | 0C0BDA4FB | 3BF4B94FEF | 1 | 1 | 1 | 1 | 1 | 00 | 01 | 01 |
| 273 | 34 | 1A7E56F | 7BA038DC | 1817B49F6 | 77E9729FDE | 1 | 1 | 1 | 1 | 0 | 01 | 00 | 01 |
| 274 | 35 | 14FCADF | 774071B9 | 102F693ED | 6FD2E53FBD | 0 | 0 | 0 | 1 | 0 | 00 | 01 | 00 |
| 275 | 36 | 09F95BE | 6E80E373 | 005ED27DB | 5FA5CA7F7B | 1 | 1 | 0 | 1 | 1 | 10 | 00 | 01 |
| 276 | 37 | 13F2B7C | 5D01C6E7 | 00BDA4FB6 | 3F4B94FEF7 | 0 | 0 | 0 | 0 | 0 | 11 | 10 | 00 |
| 277 | 38 | 07E56F9 | 3A038DCE | 017B49F6C | 7E9729FDEE | 0 | 0 | 0 | 1 | 0 | 00 | 11 | 10 |
| 278 | 39 | 0FCADF2 | 74071B9C | 02F693ED8 | 7D2E53FBDD | 1 | 0 | 0 | 0 | 1 | 10 | 00 | 11 |
| 279 | 40 | 1F95BE5 | 680E3738 | 05ED27DB0 | 7A5CA7F7BA | 1 | 0 | 0 | 0 | 1 | 11 | 10 | 00 |
| 280 | 41 | 1F2B7CA | 501C6E71 | 0BDA4FB60 | 74B94FEF74 | 1 | 0 | 1 | 1 | 0 | 01 | 11 | 10 |
| 281 | 42 | 1E56F94 | 2038DCE2 | 17B49F6C0 | 69729FDEE8 | 1 | 0 | 0 | 0 | 0 | 10 | 01 | 11 |
| 282 | 43 | 1CADF29 | 4071B9C4 | 0F693ED80 | 52E53FBDD1 | 1 | 0 | 1 | 1 | 1 | 11 | 10 | 01 |
| 283 | 44 | 195BE53 | 00E37389 | 1ED27DB01 | 25CA7F7BA3 | 1 | 1 | 1 | 1 | 1 | 01 | 11 | 10 |
| 284 | 45 | 12B7CA6 | 01C6E713 | 1DA4FB602 | 4B94FEF747 | 0 | 1 | 1 | 1 | 0 | 01 | 01 | 11 |
| 285 | 46 | 056F94C | 038DCE26 | 1B49F6C04 | 1729FDEE8E | 0 | 1 | 1 | 0 | 1 | 11 | 01 | 01 |
| 286 | 47 | 0ADF299 | 071B9C4D | 1693ED808 | 2E53FBDD1C | 1 | 0 | 0 | 0 | 0 | 10 | 11 | 01 |
| 287 | 48 | 15BE532 | 0E37389A | 0D27DB011 | 5CA7F7BA38 | 0 | 0 | 1 | 1 | 0 | 10 | 10 | 11 |
| 288 | 49 | 0B7CA64 | 1C6E7135 | 1A4FB6022 | 394FEF7471 | 1 | 0 | 1 | 0 | 0 | 01 | 10 | 10 |
| 289 | 50 | 16F94C9 | 38DCE26A | 149F6C044 | 729FDEE8E2 | 0 | 1 | 0 | 1 | 1 | 01 | 01 | 10 |
| 290 | 51 | 0DF2993 | 71B9C4D4 | 093ED8089 | 653FBDD1C4 | 1 | 1 | 1 | 0 | 0 | 00 | 01 | 01 |
| 291 | 52 | 1BE5327 | 637389A9 | 127DB0112 | 4A7F7BA388 | 1 | 0 | 0 | 0 | 1 | 11 | 00 | 01 |
| 292 | 53 | 17CA64E | 46E71353 | 04FB60224 | 14FEF74710 | 0 | 1 | 0 | 1 | 1 | 01 | 11 | 00 |
| 293 | 54 | 0F94C9C | 0DCE26A6 | 09F6C0448 | 29FDEE8E21 | 1 | 1 | 1 | 1 | 1 | 01 | 01 | 11 |
| 294 | 55 | 1F29939 | 1B9C4D4D | 13ED80890 | 53FBDD1C42 | 1 | 1 | 0 | 1 | 0 | 00 | 01 | 01 |
| 295 | 56 | 1E53272 | 37389A9A | 07DB01121 | 27F7BA3884 | 1 | 0 | 0 | 1 | 0 | 10 | 00 | 01 |
| 296 | 57 | 1CA64E5 | 6E713534 | 0FB602242 | 4FEF747108 | 1 | 0 | 1 | 1 | 1 | 00 | 10 | 00 |
| 297 | 58 | 194C9CB | 5CE26A69 | 1F6C04485 | 1FDEE8E210 | 1 | 1 | 1 | 1 | 0 | 11 | 00 | 10 |
| 298 | 59 | 1299397 | 39C4D4D3 | 1ED80890A | 3FBDD1C420 | 0 | 1 | 1 | 1 | 0 | 00 | 11 | 00 |
| 299 | 60 | 053272F | 7389A9A6 | 1DB011214 | 7F7BA38840 | 0 | 1 | 1 | 0 | 0 | 11 | 00 | 11 |
| 300 | 61 | 0A64E5E | 6713534C | 1B6022428 | 7EF7471081 | 1 | 0 | 1 | 1 | 0 | 00 | 11 | 00 |
| 301 | 62 | 14C9CBD | 4E26A699 | 16C044850 | 7DEE8E2102 | 0 | 0 | 0 | 1 | 1 | 10 | 00 | 11 |
| 302 | 63 | 099397A | 1C4D4D32 | 0D80890A0 | 7BDD1C4205 | 1 | 0 | 1 | 1 | 1 | 00 | 10 | 00 |
| 303 | 64 | 13272F4 | 389A9A65 | 1B0112141 | 77BA38840B | 0 | 1 | 1 | 1 | 1 | 00 | 00 | 10 |
| 304 | 65 | 064E5E8 | 713534CB | 160224283 | 6F74710817 | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 305 | 66 | 0C9CBD1 | 626A6997 | 0C0448507 | 5EE8E2102E | 1 | 0 | 1 | 1 | 1 | 01 | 00 | 00 |
| 306 | 67 | 19397A3 | 44D4D32E | 180890A0E | 3DD1C4205C | 1 | 1 | 1 | 1 | 1 | 11 | 01 | 00 |
| 307 | 68 | 1272F46 | 09A9A65D | 10112141D | 7BA38840B8 | 0 | 1 | 0 | 1 | 1 | 10 | 11 | 01 |
| 308 | 69 | 04E5E8C | 13534CBA | 00224283A | 7747108171 | 0 | 0 | 0 | 0 | 0 | 01 | 10 | 11 |
| 309 | 70 | 09CBD19 | 26A69975 | 004485075 | 6E8E2102E3 | 1 | 1 | 0 | 1 | 0 | 10 | 01 | 10 |
| 310 | 71 | 1397A32 | 4D4D32EB | 00890A0EA | 5D1C4205C7 | 0 | 0 | 0 | 0 | 0 | 00 | 10 | 01 |
| 311 | 72 | 072F465 | 1A9A65D7 | 0112141D5 | 3A38840B8F | 0 | 1 | 0 | 0 | 1 | 01 | 00 | 10 |
| 312 | 73 | 0E5E8CA | 3534CBAF | 0224283AA | 747108171F | 1 | 0 | 0 | 0 | 0 | 00 | 01 | 00 |
| 313 | 74 | 1CBD194 | 6A69975E | 044850755 | 68E2102E3E | 1 | 0 | 0 | 1 | 0 | 10 | 00 | 01 |
| 314 | 75 | 197A329 | 54D32EBC | 0890A0EAB | 51C4205C7D | 1 | 1 | 1 | 1 | 0 | 01 | 10 | 00 |

*Sample Data*

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 315 | 76 | 12F4653 | 29A65D79 | 112141D56 | 238840B8FA | 0 | 1 | 0 | 1 | 1 | 01 | 01 | 10 |
| 316 | 77 | 05E8CA6 | 534CBAF2 | 024283AAD | 47108171F4 | 0 | 0 | 0 | 0 | 1 | 10 | 01 | 01 |
| 317 | 78 | 0BD194D | 269975E5 | 04850755B | 0E2102E3E9 | 1 | 1 | 0 | 0 | 0 | 11 | 10 | 01 |
| 318 | 79 | 17A329A | 4D32EBCB | 090A0EAB6 | 1C4205C7D2 | 0 | 0 | 1 | 0 | 0 | 00 | 11 | 10 |
| 319 | 80 | 0F46535 | 1A65D797 | 12141D56D | 38840B8FA5 | 1 | 0 | 0 | 1 | 0 | 11 | 00 | 11 |
| 320 | 81 | 1E8CA6A | 34CBAF2F | 04283AADA | 7108171F4B | 1 | 1 | 0 | 0 | 1 | 01 | 11 | 00 |
| 321 | 82 | 1D194D5 | 69975E5F | 0850755B4 | 62102E3E97 | 1 | 1 | 1 | 0 | 0 | 01 | 01 | 11 |
| 322 | 83 | 1A329AA | 532EBCBF | 10A0EAB68 | 44205C7D2F | 1 | 0 | 0 | 0 | 0 | 11 | 01 | 01 |
| 323 | 84 | 1465355 | 265D797F | 0141D56D1 | 0840B8FA5E | 0 | 0 | 0 | 0 | 1 | 01 | 11 | 01 |
| 324 | 85 | 08CA6AB | 4CBAF2FF | 0283AADA2 | 108171F4BC | 1 | 1 | 0 | 1 | 0 | 01 | 01 | 11 |
| 325 | 86 | 1194D56 | 1975E5FF | 050755B45 | 2102E3E979 | 0 | 0 | 0 | 0 | 1 | 10 | 01 | 01 |
| 326 | 87 | 0329AAD | 32EBCBFF | 0A0EAB68A | 4205C7D2F3 | 0 | 1 | 1 | 0 | 0 | 11 | 10 | 01 |
| 327 | 88 | 065355A | 65D797FF | 141D56D14 | 040B8FA5E7 | 0 | 1 | 0 | 0 | 0 | 00 | 11 | 10 |
| 328 | 89 | 0CA6AB4 | 4BAF2FFF | 083AADA28 | 08171F4BCF | 1 | 1 | 1 | 0 | 1 | 11 | 00 | 11 |
| 329 | 90 | 194D569 | 175E5FFF | 10755B450 | 102E3E979E | 1 | 0 | 0 | 0 | 0 | 01 | 11 | 00 |
| 330 | 91 | 129AAD3 | 2EBCBFFF | 00EAB68A1 | 205C7D2F3C | 0 | 1 | 0 | 0 | 0 | 10 | 01 | 11 |
| 331 | 92 | 05355A6 | 5D797FFF | 01D56D142 | 40B8FA5E78 | 0 | 0 | 0 | 1 | 1 | 00 | 10 | 01 |
| 332 | 93 | 0A6AB4D | 3AF2FFFE | 03AADA285 | 0171F4BCF1 | 1 | 1 | 0 | 0 | 0 | 00 | 00 | 10 |
| 333 | 94 | 14D569B | 75E5FFFD | 0755B450A | 02E3E979E2 | 0 | 1 | 0 | 1 | 0 | 01 | 00 | 00 |
| 334 | 95 | 09AAD37 | 6BCBFFFA | 0EAB68A15 | 05C7D2F3C4 | 1 | 1 | 1 | 1 | 1 | 11 | 01 | 00 |
| 335 | 96 | 1355A6E | 5797FFF4 | 1D56D142A | 0B8FA5E788 | 0 | 1 | 1 | 1 | 0 | 11 | 11 | 01 |
| 336 | 97 | 06AB4DC | 2F2FFFE8 | 1AADA2854 | 171F4BCF11 | 0 | 0 | 1 | 0 | 0 | 11 | 11 | 11 |
| 337 | 98 | 0D569B8 | 5E5FFFD0 | 155B450A9 | 2E3E979E23 | 1 | 0 | 0 | 0 | 0 | 11 | 11 | 11 |
| 338 | 99 | 1AAD370 | 3CBFFFA1 | 0AB68A153 | 5C7D2F3C46 | 1 | 1 | 1 | 0 | 0 | 10 | 11 | 11 |
| 339 | 100 | 155A6E0 | 797FFF43 | 156D142A7 | 38FA5E788D | 0 | 0 | 0 | 1 | 1 | 01 | 10 | 11 |
| 340 | 101 | 0AB4DC0 | 72FFFE87 | 0ADA2854E | 71F4BCF11B | 1 | 1 | 1 | 1 | 1 | 10 | 01 | 10 |
| 341 | 102 | 1569B81 | 65FFFD0E | 15B450A9D | 63E979E236 | 0 | 1 | 0 | 1 | 0 | 11 | 10 | 01 |
| 342 | 103 | 0AD3703 | 4BFFFA1C | 0B68A153B | 47D2F3C46C | 1 | 1 | 1 | 1 | 1 | 01 | 11 | 10 |
| 343 | 104 | 15A6E07 | 17FFF438 | 16D142A76 | 0FA5E788D8 | 0 | 1 | 0 | 1 | 1 | 10 | 01 | 11 |
| 344 | 105 | 0B4DC0F | 2FFFE870 | 0DA2854EC | 1F4BCF11B0 | 1 | 1 | 1 | 0 | 1 | 11 | 10 | 01 |
| 345 | 106 | 169B81F | 5FFFD0E1 | 1B450A9D8 | 3E979E2360 | 0 | 1 | 1 | 1 | 0 | 01 | 11 | 10 |
| 346 | 107 | 0D3703F | 3FFFA1C3 | 168A153B0 | 7D2F3C46C1 | 1 | 1 | 0 | 0 | 1 | 10 | 01 | 11 |
| 347 | 108 | 1A6E07E | 7FFF4386 | 0D142A761 | 7A5E788D83 | 1 | 1 | 1 | 0 | 1 | 11 | 10 | 01 |
| 348 | 109 | 14DC0FD | 7FFE870C | 1A2854EC2 | 74BCF11B07 | 0 | 1 | 1 | 1 | 0 | 01 | 11 | 10 |
| 349 | 110 | 09B81FB | 7FFD0E19 | 1450A9D84 | 6979E2360E | 1 | 1 | 0 | 0 | 1 | 10 | 01 | 11 |
| 350 | 111 | 13703F6 | 7FFA1C33 | 08A153B09 | 52F3C46C1C | 0 | 1 | 1 | 1 | 1 | 11 | 10 | 01 |
| 351 | 112 | 06E07EC | 7FF43867 | 1142A7612 | 25E788D838 | 0 | 1 | 0 | 1 | 1 | 00 | 11 | 10 |
| 352 | 113 | 0DC0FD8 | 7FE870CF | 02854EC25 | 4BCF11B071 | 1 | 1 | 0 | 1 | 1 | 11 | 00 | 11 |
| 353 | 114 | 1B81FB1 | 7FD0E19E | 050A9D84B | 179E2360E3 | 1 | 1 | 0 | 1 | 0 | 00 | 11 | 00 |
| 354 | 115 | 1703F62 | 7FA1C33D | 0A153B096 | 2F3C46C1C7 | 0 | 1 | 1 | 0 | 0 | 11 | 00 | 11 |
| 355 | 116 | 0E07EC4 | 7F43867B | 142A7612C | 5E788D838E | 1 | 0 | 0 | 0 | 0 | 01 | 11 | 00 |
| 356 | 117 | 1C0FD88 | 7E870CF6 | 0854EC259 | 3CF11B071C | 1 | 1 | 1 | 1 | 1 | 01 | 01 | 11 |
| 357 | 118 | 181FB11 | 7D0E19ED | 10A9D84B3 | 79E2360E38 | 1 | 0 | 0 | 1 | 1 | 11 | 01 | 01 |
| 358 | 119 | 103F622 | 7A1C33DA | 0153B0967 | 73C46C1C71 | 0 | 0 | 0 | 1 | 0 | 10 | 11 | 01 |
| 359 | 120 | 007EC45 | 743867B5 | 02A7612CE | 6788D838E3 | 0 | 0 | 0 | 1 | 1 | 01 | 10 | 11 |
| 360 | 121 | 00FD88B | 6870CF6B | 054EC259C | 4F11B071C6 | 0 | 0 | 0 | 0 | 1 | 00 | 01 | 10 |
| 361 | 122 | 01FB117 | 50E19ED7 | 0A9D84B38 | 1E2360E38C | 0 | 1 | 1 | 0 | 0 | 10 | 00 | 01 |
| 362 | 123 | 03F622F | 21C33DAE | 153B09671 | 3C46C1C718 | 0 | 1 | 0 | 0 | 1 | 11 | 10 | 00 |
| 363 | 124 | 07EC45F | 43867B5C | 0A7612CE2 | 788D838E30 | 0 | 1 | 1 | 1 | 0 | 01 | 11 | 10 |
| 364 | 125 | 0FD88BF | 070CF6B9 | 14EC259C4 | 711B071C61 | 1 | 0 | 0 | 0 | 0 | 10 | 01 | 11 |

## 1.3 SECOND SET OF SAMPLE DATA

```
Initial values for the key, BD_ADDR and clock

K'c2[0]   = 00   K'c2[1]   = 00    K'c2[2]   = 00    K'c2[3]  = 00
K'c2[4]   = 00   K'c2[5]   = 00    K'c2[6]   = 00    K'c2[7]  = 00
K'c2[8]   = 00   K'c2[9]   = 00    K'c2[10]  = 00    K'c2[11] = 00
K'c2[12]  = 00   K'c2[13]  = 00    K'c2[14]  = 00    K'c2[15] = 00

Addr2[0] = 00   Addr2[1] = 00    Addr2[2] = 00
Addr2[3] = 00   Addr2[4] = 00    Addr2[5] = 00

CL2[0]  = 00   CL2[1]  = 00    CL2[2]  = 00    CL2[3]  = 03


================================================================================================
         Fill LFSRs with initial data
================================================================================================

  t  clk#    LFSR1    LFSR2     LFSR3      LFSR4          X1 X2 X3 X4   Z  C[t+1] C[t] C[t-1]

  0     0    0000000* 00000000* 000000000* 0000000000*    0  0  0  0    0  00     00   00
  1     1    0000001* 00000001* 000000001* 0000000001*    0  0  0  0    0  00     00   00
  2     2    0000002* 00000002* 000000002* 0000000003*    0  0  0  0    0  00     00   00
  3     3    0000004* 00000004* 000000004* 0000000007*    0  0  0  0    0  00     00   00
  4     4    0000008* 00000008* 000000008* 000000000E*    0  0  0  0    0  00     00   00
  5     5    0000010* 00000010* 000000010* 000000001C*    0  0  0  0    0  00     00   00
  6     6    0000020* 00000020* 000000020* 0000000038*    0  0  0  0    0  00     00   00
  7     7    0000040* 00000040* 000000040* 0000000070*    0  0  0  0    0  00     00   00
  8     8    0000080* 00000080* 000000080* 00000000E0*    0  0  0  0    0  00     00   00
  9     9    0000100* 00000100* 000000100* 00000001C0*    0  0  0  0    0  00     00   00
 10    10    0000200* 00000200* 000000200* 0000000380*    0  0  0  0    0  00     00   00
 11    11    0000400* 00000400* 000000400* 0000000700*    0  0  0  0    0  00     00   00
 12    12    0000800* 00000800* 000000800* 0000000E00*    0  0  0  0    0  00     00   00
 13    13    0001000* 00001000* 000001000* 0000001C00*    0  0  0  0    0  00     00   00
 14    14    0002000* 00002000* 000002000* 0000003800*    0  0  0  0    0  00     00   00
 15    15    0004000* 00004000* 000004000* 0000007000*    0  0  0  0    0  00     00   00
 16    16    0008000* 00008000* 000008000* 000000E000*    0  0  0  0    0  00     00   00
 17    17    0010000* 00010000* 000010000* 000001C000*    0  0  0  0    0  00     00   00
 18    18    0020000* 00020000* 000020000* 0000038000*    0  0  0  0    0  00     00   00
 19    19    0040000* 00040000* 000040000* 0000070000*    0  0  0  0    0  00     00   00
 20    20    0080000* 00080000* 000080000* 00000E0000*    0  0  0  0    0  00     00   00
 21    21    0100000* 00100000* 000100000* 00001C0000*    0  0  0  0    0  00     00   00
 22    22    0200000* 00200000* 000200000* 0000380000*    0  0  0  0    0  00     00   00
 23    23    0400000* 00400000* 000400000* 0000700000*    0  0  0  0    0  00     00   00
 24    24    0800000* 00800000* 000800000* 0000E00000*    1  1  0  0    0  01     00   00
 25    25    1000000* 01000000* 001000000* 0001C00000*    0  0  0  0    0  00     00   00
 26    26    0000001  02000000* 002000000* 0003800000*    0  0  0  0    0  00     00   00
 27    27    0000002  04000000* 004000000* 0007000000*    0  0  0  0    0  00     00   00
 28    28    0000004  08000000* 008000000* 000E000000*    0  0  0  0    0  00     00   00
 29    29    0000008  10000000* 010000000* 001C000000*    0  0  0  0    0  00     00   00
 30    30    0000010  20000000* 020000000* 0038000000*    0  0  0  0    0  00     00   00
 31    31    0000020  40000000* 040000000* 0070000000*    0  0  0  0    0  00     00   00
 32    32    0000040  00000001  080000000* 00E0000000*    0  0  1  1    0  01     00   00
 33    33    0000080  00000002  100000000* 01C0000000*    0  0  0  1    1  00     00   00
 34    34    0000101  00000004  000000001  0380000000*    0  0  0  1    1  00     00   00
```

*Sample Data*

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 35 | 35 | 0000202 | 00000008 | 000000002 | 0700000000* | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 36 | 36 | 0000404 | 00000010 | 000000004 | 0E00000000* | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 37 | 37 | 0000808 | 00000020 | 000000008 | 1C00000000* | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 38 | 38 | 0001011 | 00000040 | 000000011 | 3800000000* | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 39 | 39 | 0002022 | 00000080 | 000000022 | 7000000000* | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |

Start clocking Summation Combiner

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 40 | 1 | 0004044 | 00000100 | 000000044 | 6000000001 | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 41 | 2 | 0008088 | 00000200 | 000000088 | 4000000003 | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 42 | 3 | 0010111 | 00000400 | 000000111 | 0000000007 | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 43 | 4 | 0020222 | 00000800 | 000000222 | 000000000E | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 44 | 5 | 0040444 | 00001001 | 000000444 | 000000001D | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 45 | 6 | 0080888 | 00002002 | 000000888 | 000000003B | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 46 | 7 | 0101111 | 00004004 | 000001111 | 0000000077 | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 47 | 8 | 0202222 | 00008008 | 000002222 | 00000000EE | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 48 | 9 | 0404444 | 00010011 | 000004444 | 00000001DD | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 49 | 10 | 0808888 | 00020022 | 000008888 | 00000003BB | 1 | 0 | 0 | 0 | 1 | 00 | 00 | 00 |
| 50 | 11 | 1011110 | 00040044 | 000011111 | 0000000777 | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 51 | 12 | 0022221 | 00080088 | 000022222 | 0000000EEE | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 52 | 13 | 0044442 | 00100110 | 000044444 | 0000001DDD | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 53 | 14 | 0088884 | 00200220 | 000088888 | 0000003BBB | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 54 | 15 | 0111109 | 00400440 | 000111111 | 0000007777 | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 55 | 16 | 0222212 | 00800880 | 000222222 | 000000EEEE | 0 | 1 | 0 | 0 | 1 | 00 | 00 | 00 |
| 56 | 17 | 0444424 | 01001100 | 000444444 | 000001DDDD | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 57 | 18 | 0888848 | 02002200 | 000888888 | 000003BBBB | 1 | 0 | 0 | 0 | 1 | 00 | 00 | 00 |
| 58 | 19 | 1111090 | 04004400 | 001111110 | 0000077777 | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 59 | 20 | 0222120 | 08008800 | 002222220 | 00000EEEEE | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 60 | 21 | 0444240 | 10011000 | 004444440 | 00001DDDDD | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 61 | 22 | 0888480 | 20022000 | 008888880 | 00003BBBBB | 1 | 0 | 0 | 0 | 1 | 00 | 00 | 00 |
| 62 | 23 | 1110900 | 40044000 | 011111100 | 0000777777 | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 63 | 24 | 0221200 | 00088001 | 022222200 | 0000EEEEEE | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 64 | 25 | 0442400 | 00110003 | 044444400 | 0001DDDDDD | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 65 | 26 | 0884800 | 00220006 | 088888800 | 0003BBBBBB | 1 | 0 | 1 | 0 | 0 | 01 | 00 | 00 |
| 66 | 27 | 1109000 | 0044000C | 111111000 | 0007777777 | 0 | 0 | 0 | 0 | 1 | 01 | 01 | 00 |
| 67 | 28 | 0212001 | 00880018 | 022222001 | 000EEEEEEE | 0 | 1 | 0 | 0 | 0 | 11 | 01 | 01 |
| 68 | 29 | 0424002 | 01100031 | 044444002 | 001DDDDDDC | 0 | 0 | 0 | 0 | 1 | 01 | 11 | 01 |
| 69 | 30 | 0848004 | 02200062 | 088888004 | 003BBBBBB8 | 1 | 0 | 1 | 0 | 1 | 10 | 01 | 11 |
| 70 | 31 | 1090008 | 044000C4 | 111110008 | 0077777770 | 0 | 0 | 0 | 0 | 0 | 00 | 10 | 01 |
| 71 | 32 | 0120010 | 08800188 | 022220010 | 00EEEEEEE0 | 0 | 1 | 0 | 1 | 0 | 00 | 00 | 10 |
| 72 | 33 | 0240020 | 11000311 | 044440020 | 01DDDDDDC1 | 0 | 0 | 0 | 1 | 1 | 00 | 00 | 00 |
| 73 | 34 | 0480040 | 22000622 | 088880040 | 03BBBBBB83 | 0 | 0 | 1 | 1 | 0 | 01 | 00 | 00 |
| 74 | 35 | 0900081 | 44000C44 | 111100080 | 0777777707 | 1 | 0 | 0 | 0 | 0 | 00 | 01 | 00 |
| 75 | 36 | 1200103 | 08001888 | 022200101 | 0EEEEEEE0E | 0 | 0 | 0 | 1 | 1 | 11 | 00 | 01 |
| 76 | 37 | 0400207 | 10003111 | 044400202 | 1DDDDDDC1D | 0 | 0 | 0 | 1 | 0 | 01 | 11 | 00 |
| 77 | 38 | 080040E | 20006222 | 088800404 | 3BBBBBB83B | 1 | 0 | 1 | 1 | 0 | 01 | 01 | 11 |
| 78 | 39 | 100081C | 4000C444 | 111000808 | 7777777077 | 0 | 0 | 0 | 0 | 1 | 10 | 01 | 01 |
| 79 | 40 | 0001038 | 00018888 | 022001010 | 6EEEEEE0EF | 0 | 0 | 0 | 1 | 1 | 00 | 10 | 01 |
| 80 | 41 | 0002070 | 00031110 | 044002020 | 5DDDDDC1DE | 0 | 0 | 0 | 1 | 1 | 01 | 00 | 10 |
| 81 | 42 | 00040E0 | 00062220 | 088004040 | 3BBBBB83BC | 0 | 0 | 1 | 1 | 1 | 00 | 01 | 00 |
| 82 | 43 | 00081C1 | 000C4440 | 110008081 | 7777770779 | 0 | 0 | 0 | 0 | 0 | 11 | 00 | 01 |
| 83 | 44 | 0010383 | 00188880 | 020010103 | 6EEEEE0EF2 | 0 | 0 | 0 | 1 | 0 | 01 | 11 | 00 |
| 84 | 45 | 0020707 | 00311100 | 040020206 | 5DDDDC1DE5 | 0 | 0 | 0 | 1 | 0 | 10 | 01 | 11 |
| 85 | 46 | 0040E0E | 00622200 | 08004040C | 3BBBB83BCB | 0 | 0 | 1 | 1 | 0 | 11 | 10 | 01 |
| 86 | 47 | 0081C1D | 00C44400 | 100080819 | 7777707797 | 0 | 1 | 0 | 0 | 0 | 00 | 11 | 10 |
| 87 | 48 | 010383A | 01888801 | 000101032 | 6EEEE0EF2F | 0 | 1 | 0 | 1 | 0 | 11 | 00 | 11 |
| 88 | 49 | 0207075 | 03111003 | 000202064 | 5DDDC1DE5E | 0 | 0 | 0 | 1 | 0 | 01 | 11 | 00 |

*Sample Data*

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 89 | 50 | 040E0EA | 06222006 | 0004040C8 | 3BBB83BCBC | 0 | 0 | 0 | 1 | 0 | 10 | 01 | 11 |
| 90 | 51 | 081C1D5 | 0C44400C | 000808191 | 7777077979 | 1 | 0 | 0 | 0 | 1 | 00 | 10 | 01 |
| 91 | 52 | 10383AB | 18888018 | 001010323 | 6EEE0EF2F2 | 0 | 1 | 0 | 1 | 0 | 00 | 00 | 10 |
| 92 | 53 | 0070756 | 31110030 | 002020646 | 5DDC1DE5E5 | 0 | 0 | 0 | 1 | 1 | 00 | 00 | 00 |
| 93 | 54 | 00E0EAC | 62220060 | 004040C8C | 3BB83BCBCB | 0 | 0 | 0 | 1 | 1 | 00 | 00 | 00 |
| 94 | 55 | 01C1D59 | 444400C1 | 008081919 | 7770779797 | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 95 | 56 | 0383AB2 | 08880183 | 010103232 | 6EE0EF2F2F | 0 | 1 | 0 | 1 | 0 | 01 | 00 | 00 |
| 96 | 57 | 0707565 | 11100307 | 020206464 | 5DC1DE5E5F | 0 | 0 | 0 | 1 | 0 | 00 | 01 | 00 |
| 97 | 58 | 0E0EACA | 2220060E | 04040C8C8 | 3B83BCBCBF | 1 | 0 | 0 | 1 | 0 | 10 | 00 | 01 |
| 98 | 59 | 1C1D594 | 44400C1C | 080819191 | 770779797E | 1 | 0 | 1 | 0 | 0 | 00 | 10 | 00 |
| 99 | 60 | 183AB28 | 08801838 | 101032323 | 6E0EF2F2FC | 1 | 1 | 0 | 0 | 0 | 00 | 00 | 10 |
| 100 | 61 | 1075650 | 11003070 | 002064647 | 5C1DE5E5F8 | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 101 | 62 | 00EACA1 | 220060E0 | 0040C8C8E | 383BCBCBF0 | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 102 | 63 | 01D5943 | 4400C1C0 | 00819191D | 70779797E0 | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 103 | 64 | 03AB286 | 08018380 | 01032323A | 60EF2F2FC1 | 0 | 0 | 0 | 1 | 1 | 00 | 00 | 00 |
| 104 | 65 | 075650C | 10030701 | 020646475 | 41DE5E5F82 | 0 | 0 | 0 | 1 | 1 | 00 | 00 | 00 |
| 105 | 66 | 0EACA18 | 20060E02 | 040C8C8EA | 03BCBCBF04 | 1 | 0 | 0 | 1 | 0 | 01 | 00 | 00 |
| 106 | 67 | 1D59430 | 400C1C05 | 0819191D4 | 0779797E09 | 1 | 0 | 1 | 0 | 1 | 00 | 01 | 00 |
| 107 | 68 | 1AB2861 | 0018380A | 1032323A9 | 0EF2F2FC12 | 1 | 0 | 0 | 1 | 0 | 10 | 00 | 01 |
| 108 | 69 | 15650C3 | 00307015 | 006464752 | 1DE5E5F825 | 0 | 0 | 0 | 1 | 1 | 11 | 10 | 00 |
| 109 | 70 | 0ACA186 | 0060E02A | 00C8C8EA4 | 3BCBCBF04B | 1 | 0 | 0 | 1 | 1 | 00 | 11 | 10 |
| 110 | 71 | 159430C | 00C1C055 | 019191D48 | 779797E097 | 0 | 1 | 0 | 1 | 0 | 11 | 00 | 11 |
| 111 | 72 | 0B28618 | 018380AA | 032323A90 | 6F2F2FC12F | 1 | 1 | 0 | 0 | 1 | 01 | 11 | 00 |
| 112 | 73 | 1650C30 | 03070154 | 064647520 | 5E5E5F825E | 0 | 0 | 0 | 0 | 1 | 11 | 01 | 11 |
| 113 | 74 | 0CA1860 | 060E02A8 | 0C8C8EA40 | 3CBCBF04BC | 1 | 0 | 1 | 1 | 0 | 11 | 11 | 01 |
| 114 | 75 | 19430C0 | 0C1C0550 | 19191D480 | 79797E0979 | 1 | 0 | 1 | 0 | 1 | 11 | 11 | 11 |
| 115 | 76 | 1286180 | 18380AA0 | 12323A900 | 72F2FC12F2 | 0 | 0 | 0 | 1 | 0 | 11 | 11 | 11 |
| 116 | 77 | 050C301 | 30701541 | 046475201 | 65E5F825E5 | 0 | 0 | 0 | 1 | 0 | 11 | 11 | 11 |
| 117 | 78 | 0A18602 | 60E02A82 | 08C8EA402 | 4BCBF04BCB | 1 | 1 | 1 | 1 | 1 | 10 | 11 | 11 |
| 118 | 79 | 1430C04 | 41C05505 | 1191D4804 | 1797E09796 | 0 | 1 | 0 | 1 | 0 | 10 | 10 | 11 |
| 119 | 80 | 0861808 | 0380AA0A | 0323A9008 | 2F2FC12F2C | 1 | 1 | 0 | 0 | 0 | 01 | 10 | 10 |
| 120 | 81 | 10C3011 | 07015415 | 064752011 | 5E5F825E59 | 0 | 0 | 0 | 0 | 1 | 00 | 01 | 10 |
| 121 | 82 | 0186022 | 0E02A82A | 0C8EA4022 | 3CBF04BCB2 | 0 | 0 | 1 | 1 | 0 | 10 | 00 | 01 |
| 122 | 83 | 030C045 | 1C055054 | 191D48044 | 797E097964 | 0 | 0 | 1 | 0 | 1 | 11 | 10 | 00 |
| 123 | 84 | 061808A | 380AA0A8 | 123A90088 | 72FC12F2C9 | 0 | 0 | 0 | 1 | 0 | 00 | 11 | 10 |
| 124 | 85 | 0C30115 | 70154151 | 047520111 | 65F825E593 | 1 | 0 | 0 | 1 | 0 | 11 | 00 | 11 |
| 125 | 86 | 186022A | 602A82A3 | 08EA40222 | 4BF04BCB26 | 1 | 0 | 1 | 1 | 0 | 00 | 11 | 00 |
| 126 | 87 | 10C0455 | 40550546 | 11D480444 | 17E097964C | 0 | 0 | 0 | 1 | 1 | 10 | 00 | 11 |
| 127 | 88 | 01808AA | 00AA0A8D | 03A900888 | 2FC12F2C99 | 0 | 1 | 0 | 1 | 0 | 00 | 10 | 00 |
| 128 | 89 | 0301155 | 0154151A | 075201111 | 5F825E5932 | 0 | 0 | 0 | 1 | 1 | 01 | 00 | 10 |
| 129 | 90 | 06022AA | 02A82A34 | 0EA402222 | 3F04BCB264 | 0 | 1 | 1 | 0 | 1 | 00 | 01 | 00 |
| 130 | 91 | 0C04555 | 05505468 | 1D4804445 | 7E097964C9 | 1 | 0 | 1 | 0 | 0 | 10 | 00 | 01 |
| 131 | 92 | 1808AAA | 0AA0A8D0 | 1A900888A | 7C12F2C992 | 1 | 1 | 1 | 0 | 1 | 00 | 10 | 00 |
| 132 | 93 | 1011555 | 154151A1 | 152011115 | 7825E59324 | 0 | 0 | 0 | 0 | 0 | 01 | 00 | 10 |
| 133 | 94 | 0022AAB | 2A82A342 | 0A402222B | 704BCB2648 | 0 | 1 | 1 | 0 | 1 | 00 | 01 | 00 |
| 134 | 95 | 0045556 | 55054684 | 148044457 | 6097964C91 | 0 | 0 | 0 | 1 | 1 | 11 | 00 | 01 |
| 135 | 96 | 008AAAC | 2A0A8D09 | 0900888AE | 412F2C9923 | 0 | 0 | 1 | 0 | 0 | 01 | 11 | 00 |
| 136 | 97 | 0115559 | 54151A12 | 12011115D | 025E593246 | 0 | 0 | 0 | 0 | 1 | 11 | 01 | 11 |
| 137 | 98 | 022AAB2 | 282A3424 | 0402222BA | 04BCB2648D | 0 | 0 | 0 | 1 | 0 | 10 | 11 | 01 |
| 138 | 99 | 0455564 | 50546848 | 080444575 | 097964C91A | 0 | 0 | 1 | 0 | 1 | 01 | 10 | 11 |
| 139 | 100 | 08AAAC8 | 20A8D090 | 100888AEA | 12F2C99235 | 1 | 1 | 0 | 1 | 0 | 10 | 01 | 10 |
| 140 | 101 | 1155591 | 4151A120 | 0011115D5 | 25E593246A | 0 | 0 | 0 | 1 | 1 | 00 | 10 | 01 |
| 141 | 102 | 02AAB22 | 02A34240 | 002222BAA | 4BCB2648D5 | 0 | 1 | 0 | 1 | 0 | 00 | 00 | 10 |
| 142 | 103 | 0555644 | 05468481 | 004445755 | 17964C91AB | 0 | 0 | 0 | 1 | 1 | 00 | 00 | 00 |
| 143 | 104 | 0AAAC88 | 0A8D0903 | 00888AEAA | 2F2C992357 | 1 | 1 | 0 | 0 | 0 | 01 | 00 | 00 |
| 144 | 105 | 1555911 | 151A1206 | 011115D55 | 5E593246AE | 0 | 0 | 0 | 0 | 1 | 01 | 01 | 00 |
| 145 | 106 | 0AAB222 | 2A34240C | 02222BAAA | 3CB2648D5C | 1 | 0 | 0 | 1 | 1 | 11 | 01 | 01 |

*Sample Data*

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 146 | 107 | 1556445 | 54684818 | 044457555 | 7964C91AB8 | 0 | 0 | 0 | 0 | 1 | 01 | 11 | 01 |
| 147 | 108 | 0AAC88B | 28D09030 | 0888AEAAA | 72C9923571 | 1 | 1 | 1 | 1 | 1 | 01 | 01 | 11 |
| 148 | 109 | 1559117 | 51A12060 | 11115D555 | 6593246AE2 | 0 | 1 | 0 | 1 | 1 | 11 | 01 | 01 |
| 149 | 110 | 0AB222F | 234240C0 | 0222BAAAB | 4B2648D5C5 | 1 | 0 | 0 | 0 | 0 | 10 | 11 | 01 |
| 150 | 111 | 156445F | 46848180 | 044575557 | 164C91AB8A | 0 | 1 | 0 | 0 | 1 | 01 | 10 | 11 |
| 151 | 112 | 0AC88BF | 0D090301 | 088AEAAAE | 2C99235714 | 1 | 0 | 1 | 1 | 0 | 10 | 01 | 10 |
| 152 | 113 | 159117F | 1A120602 | 1115D555D | 593246AE28 | 0 | 0 | 0 | 0 | 0 | 00 | 10 | 01 |
| 153 | 114 | 0B222FE | 34240C04 | 022BAAABA | 32648D5C51 | 1 | 0 | 0 | 0 | 1 | 01 | 00 | 10 |
| 154 | 115 | 16445FD | 68481809 | 045755574 | 64C91AB8A2 | 0 | 0 | 0 | 1 | 0 | 00 | 01 | 00 |
| 155 | 116 | 0C88BFA | 50903012 | 08AEAAAE8 | 4992357144 | 1 | 1 | 1 | 1 | 0 | 01 | 00 | 01 |
| 156 | 117 | 19117F5 | 21206024 | 115D555D1 | 13246AE288 | 1 | 0 | 0 | 0 | 0 | 00 | 01 | 00 |
| 157 | 118 | 1222FEA | 4240C048 | 02BAAABA2 | 2648D5C511 | 0 | 0 | 0 | 0 | 0 | 11 | 00 | 01 |
| 158 | 119 | 0445FD5 | 04818090 | 057555744 | 4C91AB8A23 | 0 | 1 | 0 | 1 | 1 | 01 | 11 | 00 |
| 159 | 120 | 088BFAA | 09030120 | 0AEAAAE88 | 1923571446 | 1 | 0 | 1 | 0 | 1 | 10 | 01 | 11 |
| 160 | 121 | 1117F55 | 12060240 | 15D555D11 | 3246AE288D | 0 | 0 | 0 | 0 | 0 | 00 | 10 | 01 |
| 161 | 122 | 022FEAA | 240C0480 | 0BAAABA22 | 648D5C511B | 0 | 0 | 1 | 1 | 0 | 00 | 00 | 10 |
| 162 | 123 | 045FD54 | 48180900 | 175557444 | 491AB8A237 | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 163 | 124 | 08BFAA9 | 10301200 | 0EAAAE889 | 123571446F | 1 | 0 | 1 | 0 | 0 | 01 | 00 | 00 |
| 164 | 125 | 117F553 | 20602400 | 1D555D113 | 246AE288DF | 0 | 0 | 1 | 0 | 0 | 00 | 01 | 00 |
| 165 | 126 | 02FEAA7 | 40C04800 | 1AAABA227 | 48D5C511BE | 0 | 1 | 1 | 1 | 1 | 10 | 00 | 01 |
| 166 | 127 | 05FD54F | 01809001 | 15557444F | 11AB8A237D | 0 | 1 | 0 | 1 | 0 | 00 | 10 | 00 |
| 167 | 128 | 0BFAA9F | 03012002 | 0AAAE889E | 23571446FA | 1 | 0 | 1 | 0 | 0 | 00 | 00 | 10 |
| 168 | 129 | 17F553F | 06024004 | 1555D113D | 46AE288DF5 | 0 | 0 | 0 | 1 | 1 | 00 | 00 | 00 |
| 169 | 130 | 0FEAA7E | 0C048008 | 0AABA227A | 0D5C511BEA | 1 | 0 | 1 | 0 | 0 | 01 | 00 | 00 |
| 170 | 131 | 1FD54FC | 18090011 | 1557444F5 | 1AB8A237D5 | 1 | 0 | 0 | 1 | 1 | 00 | 01 | 00 |
| 171 | 132 | 1FAA9F9 | 30120022 | 0AAE889EB | 3571446FAA | 1 | 0 | 1 | 0 | 0 | 10 | 00 | 01 |
| 172 | 133 | 1F553F2 | 60240044 | 155D113D7 | 6AE288DF55 | 1 | 0 | 0 | 1 | 0 | 00 | 10 | 00 |
| 173 | 134 | 1EAA7E4 | 40480089 | 0ABA227AE | 55C511BEAA | 1 | 0 | 1 | 1 | 1 | 00 | 00 | 10 |
| 174 | 135 | 1D54FC9 | 00900113 | 157444F5D | 2B8A237D54 | 1 | 1 | 0 | 1 | 1 | 01 | 00 | 00 |
| 175 | 136 | 1AA9F93 | 01200227 | 0AE889EBA | 571446FAA8 | 1 | 0 | 1 | 0 | 1 | 00 | 01 | 00 |
| 176 | 137 | 1553F26 | 0240044E | 15D113D75 | 2E288DF550 | 0 | 0 | 0 | 0 | 0 | 11 | 00 | 01 |
| 177 | 138 | 0AA7E4C | 0480089C | 0BA227AEA | 5C511BEAA0 | 1 | 1 | 1 | 0 | 0 | 00 | 11 | 00 |
| 178 | 139 | 154FC98 | 09001138 | 17444F5D4 | 38A237D540 | 0 | 0 | 0 | 1 | 1 | 10 | 00 | 11 |
| 179 | 140 | 0A9F931 | 12002270 | 0E889EBA9 | 71446FAA81 | 1 | 0 | 1 | 0 | 0 | 00 | 10 | 00 |
| 180 | 141 | 153F262 | 240044E0 | 1D113D753 | 6288DF5503 | 0 | 0 | 1 | 1 | 0 | 00 | 00 | 10 |
| 181 | 142 | 0A7E4C5 | 480089C0 | 1A227AEA7 | 4511BEAA06 | 1 | 0 | 1 | 0 | 0 | 01 | 00 | 00 |
| 182 | 143 | 14FC98B | 10011381 | 1444F5D4F | 0A237D540D | 0 | 0 | 0 | 0 | 1 | 01 | 01 | 00 |
| 183 | 144 | 09F9316 | 20022702 | 0889EBA9E | 1446FAA81A | 1 | 0 | 1 | 0 | 1 | 11 | 01 | 01 |
| 184 | 145 | 13F262D | 40044E04 | 1113D753D | 288DF55035 | 0 | 0 | 0 | 1 | 0 | 10 | 11 | 01 |
| 185 | 146 | 07E4C5A | 00089C08 | 0227AEA7A | 511BEAA06A | 0 | 0 | 0 | 0 | 0 | 01 | 10 | 11 |
| 186 | 147 | 0FC98B4 | 00113810 | 044F5D4F5 | 2237D540D5 | 1 | 0 | 0 | 0 | 0 | 01 | 01 | 10 |
| 187 | 148 | 1F93169 | 00227021 | 089EBA9EB | 446FAA81AA | 1 | 0 | 1 | 0 | 1 | 11 | 01 | 01 |
| 188 | 149 | 1F262D2 | 0044E042 | 113D753D7 | 08DF550355 | 1 | 0 | 0 | 1 | 1 | 10 | 11 | 01 |
| 189 | 150 | 1E4C5A4 | 0089C085 | 027AEA7AE | 11BEAA06AA | 1 | 1 | 0 | 1 | 1 | 10 | 10 | 11 |
| 190 | 151 | 1C98B48 | 0113810A | 04F5D4F5C | 237D540D54 | 1 | 0 | 0 | 0 | 1 | 10 | 10 | 10 |
| 191 | 152 | 1931691 | 02270215 | 09EBA9EB8 | 46FAA81AA9 | 1 | 0 | 1 | 1 | 1 | 01 | 10 | 10 |
| 192 | 153 | 1262D22 | 044E042A | 13D753D71 | 0DF5503553 | 0 | 0 | 0 | 1 | 0 | 01 | 01 | 10 |
| 193 | 154 | 04C5A44 | 089C0854 | 07AEA7AE2 | 1BEAA06AA7 | 0 | 1 | 0 | 1 | 1 | 11 | 01 | 01 |
| 194 | 155 | 098B488 | 113810A8 | 0F5D4F5C4 | 37D540D54E | 1 | 0 | 1 | 1 | 0 | 11 | 11 | 01 |
| 195 | 156 | 1316910 | 22702150 | 1EBA9EB89 | 6FAA81AA9D | 0 | 0 | 1 | 1 | 1 | 11 | 11 | 11 |
| 196 | 157 | 062D220 | 44E042A0 | 1D753D712 | 5F5503553A | 0 | 1 | 1 | 0 | 1 | 11 | 11 | 11 |
| 197 | 158 | 0C5A440 | 09C08540 | 1AEA7AE25 | 3EAA06AA75 | 1 | 1 | 1 | 1 | 1 | 10 | 11 | 11 |
| 198 | 159 | 18B4880 | 13810A80 | 15D4F5C4B | 7D540D54EA | 1 | 1 | 0 | 0 | 0 | 10 | 10 | 11 |
| 199 | 160 | 1169100 | 27021500 | 0BA9EB897 | 7AA81AA9D5 | 0 | 0 | 1 | 1 | 0 | 01 | 10 | 10 |
| 200 | 161 | 02D2201 | 4E042A00 | 1753D712E | 75503553AB | 0 | 0 | 0 | 0 | 1 | 00 | 01 | 10 |
| 201 | 162 | 05A4403 | 1C085400 | 0EA7AE25C | 6AA06AA756 | 0 | 0 | 1 | 1 | 0 | 10 | 00 | 01 |
| 202 | 163 | 0B48807 | 3810A800 | 1D4F5C4B8 | 5540D54EAC | 1 | 0 | 1 | 0 | 0 | 00 | 10 | 00 |

*Sample Data*

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 203 | 164 | 169100F | 70215000 | 1A9EB8971 | 2A81AA9D58 | 0 | 0 | 1 | 1 | 0 | 00 | 00 | 10 |
| 204 | 165 | 0D2201E | 6042A001 | 153D712E3 | 5503553AB0 | 1 | 0 | 0 | 0 | 1 | 00 | 00 | 00 |
| 205 | 166 | 1A4403C | 40854002 | 0A7AE25C6 | 2A06AA7561 | 1 | 1 | 1 | 0 | 1 | 01 | 00 | 00 |
| 206 | 167 | 1488079 | 010A8004 | 14F5C4B8D | 540D54EAC3 | 0 | 0 | 0 | 0 | 1 | 01 | 01 | 00 |
| 207 | 168 | 09100F2 | 02150009 | 09EB8971B | 281AA9D586 | 1 | 0 | 1 | 0 | 1 | 11 | 01 | 01 |
| 208 | 169 | 12201E5 | 042A0012 | 13D712E37 | 503553AB0C | 0 | 0 | 0 | 0 | 1 | 01 | 11 | 01 |
| 209 | 170 | 04403CA | 08540024 | 07AE25C6E | 206AA75618 | 0 | 0 | 0 | 0 | 1 | 11 | 01 | 11 |
| 210 | 171 | 0880795 | 10A80048 | 0F5C4B8DD | 40D54EAC30 | 1 | 1 | 1 | 1 | 1 | 11 | 11 | 01 |
| 211 | 172 | 1100F2A | 21500091 | 1EB8971BA | 01AA9D5861 | 0 | 0 | 1 | 1 | 1 | 11 | 11 | 11 |
| 212 | 173 | 0201E54 | 42A00122 | 1D712E374 | 03553AB0C3 | 0 | 1 | 1 | 0 | 1 | 11 | 11 | 11 |
| 213 | 174 | 0403CA9 | 05400244 | 1AE25C6E9 | 06AA756186 | 0 | 0 | 1 | 1 | 1 | 11 | 11 | 11 |
| 214 | 175 | 0807952 | 0A800488 | 15C4B8DD3 | 0D54EAC30D | 1 | 1 | 0 | 0 | 1 | 11 | 11 | 11 |
| 215 | 176 | 100F2A5 | 15000911 | 0B8971BA6 | 1AA9D5861A | 0 | 0 | 1 | 1 | 1 | 11 | 11 | 11 |
| 216 | 177 | 001E54A | 2A001223 | 1712E374C | 3553AB0C35 | 0 | 0 | 0 | 0 | 1 | 00 | 11 | 11 |
| 217 | 178 | 003CA94 | 54002446 | 0E25C6E98 | 6AA756186A | 0 | 0 | 1 | 1 | 0 | 11 | 00 | 11 |
| 218 | 179 | 0079528 | 2800488D | 1C4B8DD31 | 554EAC30D5 | 0 | 0 | 1 | 0 | 0 | 01 | 11 | 00 |
| 219 | 180 | 00F2A50 | 5000911B | 18971BA62 | 2A9D5861AA | 0 | 0 | 1 | 1 | 1 | 10 | 01 | 11 |
| 220 | 181 | 01E54A0 | 20012236 | 112E374C4 | 553AB0C355 | 0 | 0 | 0 | 0 | 0 | 00 | 10 | 01 |
| 221 | 182 | 03CA940 | 4002446C | 025C6E988 | 2A756186AA | 0 | 0 | 0 | 0 | 0 | 01 | 00 | 10 |
| 222 | 183 | 0795280 | 000488D9 | 04B8DD310 | 54EAC30D54 | 0 | 0 | 0 | 1 | 0 | 00 | 01 | 00 |
| 223 | 184 | 0F2A500 | 000911B2 | 0971BA620 | 29D5861AA8 | 1 | 0 | 1 | 1 | 1 | 10 | 00 | 01 |
| 224 | 185 | 1E54A00 | 00122364 | 12E374C40 | 53AB0C3550 | 1 | 0 | 0 | 1 | 0 | 00 | 10 | 00 |
| 225 | 186 | 1CA9400 | 002446C8 | 05C6E9880 | 2756186AA0 | 1 | 0 | 0 | 0 | 1 | 01 | 00 | 10 |
| 226 | 187 | 1952800 | 00488D90 | 0B8DD3101 | 4EAC30D540 | 1 | 0 | 1 | 1 | 0 | 11 | 01 | 00 |
| 227 | 188 | 12A5000 | 00911B20 | 171BA6202 | 1D5861AA81 | 0 | 1 | 0 | 0 | 0 | 10 | 11 | 01 |
| 228 | 189 | 054A000 | 01223640 | 0E374C404 | 3AB0C35502 | 0 | 0 | 1 | 1 | 0 | 10 | 10 | 11 |
| 229 | 190 | 0A94000 | 02446C80 | 1C6E98808 | 756186AA05 | 1 | 0 | 1 | 0 | 0 | 01 | 10 | 10 |
| 230 | 191 | 1528001 | 0488D901 | 18DD31011 | 6AC30D540B | 0 | 1 | 1 | 1 | 0 | 10 | 01 | 10 |
| 231 | 192 | 0A50003 | 0911B203 | 11BA62023 | 55861AA817 | 1 | 0 | 0 | 1 | 0 | 11 | 10 | 01 |
| 232 | 193 | 14A0006 | 12236407 | 0374C4047 | 2B0C35502F | 0 | 0 | 0 | 0 | 1 | 11 | 11 | 10 |
| 233 | 194 | 094000C | 2446C80E | 06E98808E | 56186AA05F | 1 | 0 | 0 | 0 | 0 | 11 | 11 | 11 |
| 234 | 195 | 1280018 | 488D901C | 0DD31011D | 2C30D540BF | 0 | 1 | 1 | 0 | 1 | 11 | 11 | 11 |
| 235 | 196 | 0500030 | 111B2039 | 1BA62023A | 5861AA817E | 0 | 0 | 1 | 0 | 0 | 11 | 11 | 11 |
| 236 | 197 | 0A00060 | 22364072 | 174C40475 | 30C35502FD | 1 | 0 | 0 | 1 | 1 | 11 | 11 | 11 |
| 237 | 198 | 14000C0 | 446C80E4 | 0E98808EA | 6186AA05FB | 0 | 0 | 1 | 1 | 1 | 11 | 11 | 11 |
| 238 | 199 | 0800180 | 08D901C8 | 1D31011D5 | 430D540BF6 | 1 | 1 | 1 | 0 | 0 | 10 | 11 | 11 |
| 239 | 200 | 1000301 | 11B20391 | 1A62023AB | 061AA817EC | 0 | 1 | 1 | 0 | 0 | 10 | 10 | 11 |

```
Z[0]  = 25
Z[1]  = 45
Z[2]  = 6B
Z[3]  = 55
Z[4]  = 5F
Z[5]  = C2
Z[6]  = 20
Z[7]  = E5
Z[8]  = C4
Z[9]  = F8
Z[10] = 3A
Z[11] = F1
Z[12] = FF
Z[13] = 89
Z[14] = 02
Z[15] = 35

================================================================================================
         Reload this pattern into the LFSRs
```

*Sample Data*

```
         Hold content of Summation Combiner regs and calculate new C[t+1] and Z values
================================================================================================
LFSR1  <= 1C45F25
LFSR2  <= 7FF8C245
LFSR3  <= 1893A206B
LFSR4  <= 1A02F1E555
C[t+1] <= 10

================================================================================================
         Generating 125 key symbols (encryption/decryption sequence)
================================================================================================
240     1    1C45F25  7FF8C245  1893A206B  1A02F1E555      1  1  1  0    1  10     10   11
241     2    188BE4A  7FF1848B  1127440D7  3405E3CAAB      1  1  0  0    0  01     10   10
242     3    1117C95  7FE30917  024E881AF  680BC79557      0  1  0  0    0  01     01   10
243     4    022F92B  7FC6122F  049D1035E  50178F2AAF      0  1  0  0    0  11     01   01
244     5    045F257  7F8C245E  093A206BD  202F1E555E      0  1  1  0    1  10     11   01
245     6    08BE4AE  7F1848BC  127440D7A  405E3CAABC      1  0  0  0    1  01     10   11
246     7    117C95C  7E309178  04E881AF4  00BC795579      0  0  0  1    0  01     01   10
247     8    02F92B8  7C6122F0  09D1035E8  0178F2AAF2      0  0  1  0    0  11     01   01
248     9    05F2570  78C245E1  13A206BD0  02F1E555E5      0  1  0  1    1  10     11   01
249    10    0BE4AE1  71848BC2  07440D7A0  05E3CAABCA      1  1  0  1    1  10     10   11
250    11    17C95C3  63091784  0E881AF40  0BC7955795      0  0  1  1    0  01     10   10
251    12    0F92B87  46122F09  1D1035E80  178F2AAF2B      1  0  1  1    0  10     01   10
252    13    1F2570F  0C245E12  1A206BD01  2F1E555E56      1  0  1  0    0  11     10   01
253    14    1E4AE1F  1848BC25  1440D7A03  5E3CAABCAC      1  0  0  0    0  00     11   10
254    15    1C95C3E  3091784A  0881AF407  3C79557958      1  1  1  0    1  11     00   11
255    16    192B87D  6122F094  11035E80F  78F2AAF2B1      1  0  0  1    1  01     11   00
256    17    12570FA  4245E128  0206BD01E  71E555E562      0  0  0  1    0  10     01   11
257    18    04AE1F4  048BC250  040D7A03D  63CAABCAC5      0  1  0  1    0  11     10   01
258    19    095C3E8  091784A0  081AF407A  479557958A      1  0  1  1    0  01     11   10
259    20    12B87D1  122F0941  1035E80F4  0F2AAF2B14      0  0  0  0    1  11     01   11
260    21    0570FA3  245E1283  006BD01E9  1E555E5628      0  0  0  0    1  01     11   01
261    22    0AE1F46  48BC2506  00D7A03D2  3CAABCAC50      1  1  0  1    0  01     01   11
262    23    15C3E8C  11784A0C  01AF407A5  79557958A0      0  0  0  0    1  10     01   01
263    24    0B87D18  22F09419  035E80F4A  72AAF2B140      1  1  0  1    1  11     10   01
264    25    170FA30  45E12832  06BD01E94  6555E56280      0  1  0  0    0  00     11   10
265    26    0E1F460  0BC25065  0D7A03D28  4AABCAC501      1  1  1  1    0  00     00   11
266    27    1C3E8C0  1784A0CB  1AF407A50  1557958A03      1  1  1  0    1  01     00   00
267    28    187D181  2F094196  15E80F4A0  2AAF2B1406      1  0  0  1    1  00     01   00
268    29    10FA302  5E12832C  0BD01E941  555E56280C      0  0  1  0    1  11     00   01
269    30    01F4604  3C250658  17A03D283  2ABCAC5019      0  0  0  1    0  01     11   00
270    31    03E8C09  784A0CB0  0F407A506  557958A033      0  0  1  0    0  10     01   11
271    32    07D1812  70941960  1E80F4A0C  2AF2B14066      0  1  1  1    1  11     10   01
272    33    0FA3024  612832C1  1D01E9419  55E56280CD      1  0  1  1    0  01     11   10
273    34    1F46049  42506583  1A03D2832  2BCAC5019A      1  0  1  1    0  01     01   11
274    35    1E8C093  04A0CB07  1407A5065  57958A0335      1  1  0  1    0  00     01   01
275    36    1D18127  0941960F  080F4A0CB  2F2B14066B      1  0  1  0    0  10     00   01
276    37    1A3024F  12832C1F  101E94196  5E56280CD7      1  1  0  0    0  00     10   00
277    38    146049F  2506583E  003D2832C  3CAC5019AE      0  0  0  1    1  01     00   10
278    39    08C093E  4A0CB07D  007A50658  7958A0335D      1  0  0  0    0  00     01   00
279    40    118127C  141960FA  00F4A0CB0  72B14066BA      0  0  0  1    1  11     00   01
280    41    03024F8  2832C1F4  01E941961  656280CD74      0  0  0  0    1  10     11   00
281    42    06049F1  506583E9  03D2832C2  4AC5019AE9      0  0  0  1    1  01     10   11
282    43    0C093E2  20CB07D2  07A506585  158A0335D3      1  1  0  1    0  10     01   10
283    44    18127C5  41960FA5  0F4A0CB0B  2B14066BA7      1  1  1  0    1  11     10   01
284    45    1024F8A  032C1F4B  1E9419616  56280CD74F      0  0  1  0    0  00     11   10
285    46    0049F15  06583E97  1D2832C2C  2C5019AE9F      0  0  1  0    1  10     00   11
```

*Sample Data*

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 286 | 47 | 0093E2B | 0CB07D2F | 1A5065859 | 58A0335D3E | 0 | 1 | 1 | 1 | 1 | 00 | 10 | 00 |
| 287 | 48 | 0127C56 | 1960FA5E | 14A0CB0B2 | 314066BA7D | 0 | 0 | 0 | 0 | 0 | 01 | 00 | 10 |
| 288 | 49 | 024F8AD | 32C1F4BC | 094196164 | 6280CD74FB | 0 | 1 | 1 | 1 | 0 | 11 | 01 | 00 |
| 289 | 50 | 049F15A | 6583E978 | 12832C2C8 | 45019AE9F6 | 0 | 1 | 0 | 0 | 0 | 10 | 11 | 01 |
| 290 | 51 | 093E2B5 | 4B07D2F0 | 050658591 | 0A0335D3ED | 1 | 0 | 0 | 0 | 1 | 01 | 10 | 11 |
| 291 | 52 | 127C56B | 160FA5E0 | 0A0CB0B22 | 14066BA7DA | 0 | 0 | 1 | 0 | 0 | 01 | 01 | 10 |
| 292 | 53 | 04F8AD7 | 2C1F4BC1 | 141961645 | 280CD74FB5 | 0 | 0 | 0 | 0 | 1 | 10 | 01 | 01 |
| 293 | 54 | 09F15AF | 583E9783 | 0832C2C8A | 5019AE9F6A | 1 | 0 | 1 | 0 | 0 | 11 | 10 | 01 |
| 294 | 55 | 13E2B5E | 307D2F06 | 106585915 | 20335D3ED5 | 0 | 0 | 0 | 0 | 1 | 11 | 11 | 10 |
| 295 | 56 | 07C56BD | 60FA5E0D | 00CB0B22B | 4066BA7DAA | 0 | 1 | 0 | 0 | 0 | 11 | 11 | 11 |
| 296 | 57 | 0F8AD7A | 41F4BC1B | 019616457 | 00CD74FB54 | 1 | 1 | 0 | 1 | 0 | 10 | 11 | 11 |
| 297 | 58 | 1F15AF4 | 03E97836 | 032C2C8AF | 019AE9F6A9 | 1 | 1 | 0 | 1 | 1 | 10 | 10 | 11 |
| 298 | 59 | 1E2B5E9 | 07D2F06C | 06585915E | 0335D3ED52 | 1 | 1 | 0 | 0 | 0 | 01 | 10 | 10 |
| 299 | 60 | 1C56BD2 | 0FA5E0D8 | 0CB0B22BC | 066BA7DAA4 | 1 | 1 | 1 | 0 | 0 | 10 | 01 | 10 |
| 300 | 61 | 18AD7A5 | 1F4BC1B0 | 196164578 | 0CD74FB549 | 1 | 0 | 1 | 1 | 1 | 11 | 10 | 01 |
| 301 | 62 | 115AF4B | 3E978361 | 12C2C8AF0 | 19AE9F6A92 | 0 | 1 | 0 | 1 | 1 | 00 | 11 | 10 |
| 302 | 63 | 02B5E96 | 7D2F06C2 | 0585915E0 | 335D3ED524 | 0 | 0 | 0 | 0 | 0 | 10 | 00 | 11 |
| 303 | 64 | 056BD2D | 7A5E0D85 | 0B0B22BC1 | 66BA7DAA49 | 0 | 0 | 1 | 1 | 0 | 00 | 10 | 00 |
| 304 | 65 | 0AD7A5B | 74BC1B0A | 161645783 | 4D74FB5493 | 1 | 1 | 0 | 0 | 0 | 00 | 00 | 10 |
| 305 | 66 | 15AF4B6 | 69783615 | 0C2C8AF07 | 1AE9F6A926 | 0 | 0 | 1 | 1 | 0 | 01 | 00 | 00 |
| 306 | 67 | 0B5E96D | 52F06C2B | 185915E0F | 35D3ED524C | 1 | 1 | 1 | 1 | 1 | 11 | 01 | 00 |
| 307 | 68 | 16BD2DB | 25E0D857 | 10B22BC1F | 6BA7DAA499 | 0 | 1 | 0 | 1 | 1 | 10 | 11 | 01 |
| 308 | 69 | 0D7A5B7 | 4BC1B0AF | 01645783F | 574FB54933 | 1 | 1 | 0 | 0 | 0 | 10 | 10 | 11 |
| 309 | 70 | 1AF4B6F | 1783615F | 02C8AF07F | 2E9F6A9266 | 1 | 1 | 0 | 1 | 1 | 01 | 10 | 10 |
| 310 | 71 | 15E96DF | 2F06C2BF | 05915E0FF | 5D3ED524CC | 0 | 0 | 0 | 0 | 1 | 00 | 01 | 10 |
| 311 | 72 | 0BD2DBF | 5E0D857F | 0B22BC1FE | 3A7DAA4998 | 1 | 0 | 1 | 0 | 0 | 10 | 00 | 01 |
| 312 | 73 | 17A5B7F | 3C1B0AFE | 1645783FD | 74FB549331 | 0 | 0 | 0 | 1 | 1 | 11 | 10 | 00 |
| 313 | 74 | 0F4B6FF | 783615FD | 0C8AF07FA | 69F6A92662 | 1 | 0 | 1 | 1 | 0 | 01 | 11 | 10 |
| 314 | 75 | 1E96DFF | 706C2BFB | 1915E0FF5 | 53ED524CC4 | 1 | 0 | 1 | 1 | 0 | 01 | 01 | 11 |
| 315 | 76 | 1D2DBFE | 60D857F6 | 122BC1FEB | 27DAA49988 | 1 | 1 | 0 | 1 | 0 | 00 | 01 | 01 |
| 316 | 77 | 1A5B7FD | 41B0AFEC | 045783FD7 | 4FB5493310 | 1 | 1 | 0 | 1 | 1 | 10 | 00 | 01 |
| 317 | 78 | 14B6FFA | 03615FD8 | 08AF07FAE | 1F6A926620 | 0 | 0 | 1 | 0 | 1 | 11 | 10 | 00 |
| 318 | 79 | 096DFF4 | 06C2BFB1 | 115E0FF5D | 3ED524CC40 | 1 | 1 | 0 | 1 | 0 | 01 | 11 | 10 |
| 319 | 80 | 12DBFE8 | 0D857F63 | 02BC1FEBB | 7DAA499881 | 0 | 1 | 0 | 1 | 1 | 10 | 01 | 11 |
| 320 | 81 | 05B7FD0 | 1B0AFEC6 | 05783FD77 | 7B54933103 | 0 | 0 | 0 | 0 | 0 | 00 | 10 | 01 |
| 321 | 82 | 0B6FFA1 | 3615FD8C | 0AF07FAEF | 76A9266206 | 1 | 0 | 1 | 1 | 1 | 00 | 00 | 10 |
| 322 | 83 | 16DFF42 | 6C2BFB18 | 15E0FF5DE | 6D524CC40C | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 |
| 323 | 84 | 0DBFE85 | 5857F631 | 0BC1FEBBD | 5AA4998819 | 1 | 0 | 1 | 1 | 1 | 01 | 00 | 00 |
| 324 | 85 | 1B7FD0B | 30AFEC62 | 1783FD77A | 3549331033 | 1 | 1 | 0 | 0 | 1 | 00 | 01 | 00 |
| 325 | 86 | 16FFA16 | 615FD8C5 | 0F07FAEF5 | 6A92662067 | 0 | 0 | 1 | 1 | 0 | 10 | 00 | 01 |
| 326 | 87 | 0DFF42D | 42BFB18B | 1E0FF5DEA | 5524CC40CE | 1 | 1 | 1 | 0 | 1 | 00 | 10 | 00 |
| 327 | 88 | 1BFE85B | 057F6317 | 1C1FEBBD5 | 2A4998819C | 1 | 0 | 1 | 0 | 0 | 00 | 00 | 10 |
| 328 | 89 | 17FD0B7 | 0AFEC62E | 183FD77AA | 5493310339 | 0 | 1 | 1 | 1 | 1 | 01 | 00 | 00 |
| 329 | 90 | 0FFA16F | 15FD8C5C | 107FAEF55 | 2926620672 | 1 | 1 | 0 | 0 | 1 | 00 | 01 | 00 |
| 330 | 91 | 1FF42DF | 2BFB18B9 | 00FF5DEAA | 524CC40CE5 | 1 | 1 | 0 | 0 | 0 | 10 | 00 | 01 |
| 331 | 92 | 1FE85BF | 57F63172 | 01FEBBD55 | 24998819CA | 1 | 1 | 0 | 1 | 1 | 00 | 10 | 00 |
| 332 | 93 | 1FD0B7F | 2FEC62E4 | 03FD77AAA | 4933103394 | 1 | 1 | 0 | 0 | 0 | 00 | 00 | 10 |
| 333 | 94 | 1FA16FF | 5FD8C5C9 | 07FAEF555 | 1266206728 | 1 | 1 | 0 | 0 | 0 | 01 | 00 | 00 |
| 334 | 95 | 1F42DFF | 3FB18B93 | 0FF5DEAAA | 24CC40CE51 | 1 | 1 | 1 | 1 | 1 | 11 | 01 | 00 |
| 335 | 96 | 1E85BFF | 7F631727 | 1FEBBD554 | 4998819CA3 | 1 | 0 | 1 | 1 | 0 | 11 | 11 | 01 |
| 336 | 97 | 1D0B7FE | 7EC62E4F | 1FD77AAA9 | 1331033947 | 1 | 1 | 1 | 0 | 0 | 10 | 11 | 11 |
| 337 | 98 | 1A16FFC | 7D8C5C9F | 1FAEF5553 | 266206728E | 1 | 1 | 1 | 0 | 1 | 10 | 10 | 11 |
| 338 | 99 | 142DFF9 | 7B18B93F | 1F5DEAAA7 | 4CC40CE51D | 0 | 0 | 1 | 1 | 0 | 01 | 10 | 10 |
| 339 | 100 | 085BFF3 | 7631727F | 1EBBD554E | 198819CA3B | 1 | 0 | 1 | 1 | 0 | 10 | 01 | 10 |
| 340 | 101 | 10B7FE6 | 6C62E4FF | 1D77AAA9C | 3310339477 | 0 | 0 | 1 | 0 | 1 | 00 | 10 | 01 |
| 341 | 102 | 016FFCC | 58C5C9FE | 1AEF55538 | 66206728EE | 0 | 1 | 1 | 0 | 0 | 00 | 00 | 10 |
| 342 | 103 | 02DFF98 | 318B93FC | 15DEAAA70 | 4C40CE51DC | 0 | 1 | 0 | 0 | 1 | 00 | 00 | 00 |

*Sample Data*

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 343 | 104 | 05BFF31 | 631727F8 | 0BBD554E1 | 18819CA3B9 | 0 | 0 | 1 | 1 | 0 | 01 | 00 | 00 |
| 344 | 105 | 0B7FE62 | 462E4FF1 | 177AAA9C2 | 3103394772 | 1 | 0 | 0 | 0 | 0 | 00 | 01 | 00 |
| 345 | 106 | 16FFCC5 | 0C5C9FE2 | 0EF555384 | 6206728EE4 | 0 | 0 | 1 | 0 | 1 | 11 | 00 | 01 |
| 346 | 107 | 0DFF98A | 18B93FC4 | 1DEAAA709 | 440CE51DC9 | 1 | 1 | 1 | 0 | 0 | 00 | 11 | 00 |
| 347 | 108 | 1BFF315 | 31727F88 | 1BD554E12 | 0819CA3B93 | 1 | 0 | 1 | 0 | 0 | 11 | 00 | 11 |
| 348 | 109 | 17FE62A | 62E4FF11 | 17AAA9C24 | 1033947726 | 0 | 1 | 0 | 0 | 0 | 01 | 11 | 00 |
| 349 | 110 | 0FFCC54 | 45C9FE22 | 0F5553849 | 206728EE4C | 1 | 1 | 1 | 0 | 0 | 01 | 01 | 11 |
| 350 | 111 | 1FF98A8 | 0B93FC44 | 1EAAA7093 | 40CE51DC99 | 1 | 1 | 1 | 1 | 1 | 00 | 01 | 01 |
| 351 | 112 | 1FF3150 | 1727F889 | 1D554E127 | 019CA3B933 | 1 | 0 | 1 | 1 | 1 | 10 | 00 | 01 |
| 352 | 113 | 1FE62A0 | 2E4FF112 | 1AAA9C24F | 0339477267 | 1 | 0 | 1 | 0 | 0 | 00 | 10 | 00 |
| 353 | 114 | 1FCC541 | 5C9FE225 | 15553849E | 06728EE4CF | 1 | 1 | 0 | 0 | 0 | 00 | 00 | 10 |
| 354 | 115 | 1F98A82 | 393FC44B | 0AAA7093C | 0CE51DC99F | 1 | 0 | 1 | 1 | 1 | 01 | 00 | 00 |
| 355 | 116 | 1F31504 | 727F8897 | 1554E1279 | 19CA3B933E | 1 | 0 | 0 | 1 | 1 | 00 | 01 | 00 |
| 356 | 117 | 1E62A09 | 64FF112F | 0AA9C24F2 | 339477267D | 1 | 1 | 1 | 1 | 0 | 01 | 00 | 01 |
| 357 | 118 | 1CC5412 | 49FE225E | 1553849E4 | 6728EE4CFB | 1 | 1 | 0 | 0 | 1 | 00 | 01 | 00 |
| 358 | 119 | 198A824 | 13FC44BC | 0AA7093C9 | 4E51DC99F7 | 1 | 1 | 1 | 0 | 1 | 10 | 00 | 01 |
| 359 | 120 | 1315049 | 27F88979 | 154E12792 | 1CA3B933EE | 0 | 1 | 0 | 1 | 0 | 00 | 10 | 00 |
| 360 | 121 | 062A093 | 4FF112F3 | 0A9C24F24 | 39477267DC | 0 | 1 | 1 | 0 | 0 | 00 | 00 | 10 |
| 361 | 122 | 0C54127 | 1FE225E6 | 153849E48 | 728EE4CFB8 | 1 | 1 | 0 | 1 | 1 | 01 | 00 | 00 |
| 362 | 123 | 18A824E | 3FC44BCD | 0A7093C91 | 651DC99F71 | 1 | 1 | 1 | 0 | 0 | 11 | 01 | 00 |
| 363 | 124 | 115049C | 7F88979A | 14E127922 | 4A3B933EE2 | 0 | 1 | 0 | 0 | 0 | 10 | 11 | 01 |
| 364 | 125 | 02A0938 | 7F112F35 | 09C24F244 | 1477267DC5 | 0 | 0 | 1 | 0 | 1 | 01 | 10 | 11 |

## 1.4 THIRD SET OF SAMPLES

```
Initial values for the key, pan address and clock

K'c3[0]   = FF   K'c3[1]   = FF    K'c3[2]   = FF    K'c3[3]  = FF
K'c3[4]   = FF   K'c3[5]   = FF    K'c3[6]   = FF    K'c3[7]  = FF
K'c3[8]   = FF   K'c3[9]   = FF    K'c3[10]  = FF    K'c3[11] = FF
K'c3[12]  = FF   K'c3[13]  = FF    K'c3[14]  = FF    K'c3[15] = FF

Addr3[0] = FF   Addr3[1] = FF    Addr3[2] = FF
Addr3[3] = FF   Addr3[4] = FF    Addr3[5] = FF

CL3[0]  = FF   CL3[1]  = FF    CL3[2]  = FF    CL3[3]  = 03


================================================================================================
         Fill LFSRs with initial data
================================================================================================

  t  clk#    LFSR1    LFSR2     LFSR3      LFSR4          X1 X2 X3 X4   Z  C[t+1] C[t] C[t-1]

  0     0    0000000* 00000000* 000000000* 0000000000*    0  0  0  0    0  00     00   00
  1     1    0000001* 00000001* 000000001* 0000000001*    0  0  0  0    0  00     00   00
  2     2    0000003* 00000002* 000000003* 0000000003*    0  0  0  0    0  00     00   00
  3     3    0000007* 00000004* 000000007* 0000000007*    0  0  0  0    0  00     00   00
  4     4    000000F* 00000009* 00000000F* 000000000F*    0  0  0  0    0  00     00   00
  5     5    000001F* 00000013* 00000001F* 000000001F*    0  0  0  0    0  00     00   00
  6     6    000003F* 00000027* 00000003F* 000000003F*    0  0  0  0    0  00     00   00
  7     7    000007F* 0000004F* 00000007F* 000000007F*    0  0  0  0    0  00     00   00
  8     8    00000FF* 0000009F* 0000000FF* 00000000FF*    0  0  0  0    0  00     00   00
  9     9    00001FF* 0000013F* 0000001FF* 00000001FF*    0  0  0  0    0  00     00   00
 10    10    00003FF* 0000027F* 0000003FF* 00000003FF*    0  0  0  0    0  00     00   00
 11    11    00007FF* 000004FF* 0000007FF* 00000007FF*    0  0  0  0    0  00     00   00
 12    12    0000FFF* 000009FF* 000000FFF* 0000000FFF*    0  0  0  0    0  00     00   00
 13    13    0001FFF* 000013FF* 000001FFF* 0000001FFF*    0  0  0  0    0  00     00   00
 14    14    0003FFF* 000027FF* 000003FFF* 0000003FFF*    0  0  0  0    0  00     00   00
 15    15    0007FFF* 00004FFF* 000007FFF* 0000007FFF*    0  0  0  0    0  00     00   00
 16    16    000FFFF* 00009FFF* 00000FFFF* 000000FFFF*    0  0  0  0    0  00     00   00
 17    17    001FFFF* 00013FFF* 00001FFFF* 000001FFFF*    0  0  0  0    0  00     00   00
 18    18    003FFFF* 00027FFF* 00003FFFF* 000003FFFF*    0  0  0  0    0  00     00   00
 19    19    007FFFF* 0004FFFF* 00007FFFF* 000007FFFF*    0  0  0  0    0  00     00   00
 20    20    00FFFFF* 0009FFFF* 0000FFFFF* 00000FFFFF*    0  0  0  0    0  00     00   00
 21    21    01FFFFF* 0013FFFF* 0001FFFFF* 00001FFFFF*    0  0  0  0    0  00     00   00
 22    22    03FFFFF* 0027FFFF* 0003FFFFF* 00003FFFFF*    0  0  0  0    0  00     00   00
 23    23    07FFFFF* 004FFFFF* 0007FFFFF* 00007FFFFF*    0  0  0  0    0  00     00   00
 24    24    0FFFFFF* 009FFFFF* 000FFFFFF* 0000FFFFFF*    1  1  0  0    0  01     00   00
 25    25    1FFFFFF* 013FFFFF* 001FFFFFF* 0001FFFFFF*    1  0  0  0    1  00     00   00
 26    26    1FFFFFF  027FFFFF* 003FFFFFF* 0003FFFFFF*    1  0  0  0    1  00     00   00
 27    27    1FFFFFF  04FFFFFF* 007FFFFFF* 0007FFFFFF*    1  1  0  0    0  01     00   00
 28    28    1FFFFFF  09FFFFFF* 00FFFFFFF* 000FFFFFFF*    1  1  0  0    0  01     00   00
 29    29    1FFFFFF  13FFFFFF* 01FFFFFFF* 001FFFFFFF*    1  1  0  0    0  01     00   00
 30    30    1FFFFFF  27FFFFFF* 03FFFFFFF* 003FFFFFFF*    1  1  0  0    0  01     00   00
 31    31    1FFFFFF  4FFFFFFF* 07FFFFFFF* 007FFFFFFF*    1  1  0  0    0  01     00   00
 32    32    1FFFFFF  1FFFFFFF  0FFFFFFFF* 00FFFFFFFF*    1  1  1  1    0  10     00   00
 33    33    1FFFFFF  3FFFFFFE  1FFFFFFFF* 01FFFFFFFF*    1  1  1  1    0  10     00   00
 34    34    1FFFFFF  7FFFFFFC  1FFFFFFFF  03FFFFFFF*     1  1  1  1    0  10     00   00
```

*Sample Data*

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 35 | 35 | 1FFFFF | 7FFFFF9 | 1FFFFFFF | 07FFFFFFF* | 1 | 1 | 1 | 1 | 0 | 10 | 00 | 00 |
| 36 | 36 | 1FFFFF | 7FFFFF3 | 1FFFFFFF | 0FFFFFFFF* | 1 | 1 | 1 | 1 | 0 | 10 | 00 | 00 |
| 37 | 37 | 1FFFFF | 7FFFFE7 | 1FFFFFFF | 1FFFFFFFF* | 1 | 1 | 1 | 1 | 0 | 10 | 00 | 00 |
| 38 | 38 | 1FFFFF | 7FFFFCF | 1FFFFFFF | 3FFFFFFFF* | 1 | 1 | 1 | 1 | 0 | 10 | 00 | 00 |
| 39 | 39 | 1FFFFF | 7FFFF9F | 1FFFFFFF | 7FFFFFFFF* | 1 | 1 | 1 | 1 | 0 | 10 | 00 | 00 |

Start clocking Summation Combiner

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 40 | 1 | 1FFFFF | 7FFFF3F | 1FFFFFFF | 7FFFFFFFF | 1 | 1 | 1 | 1 | 0 | 01 | 10 | 00 |
| 41 | 2 | 1FFFFF | 7FFFE7F | 1FFFFFFF | 7FFFFFFFF | 1 | 1 | 1 | 1 | 1 | 10 | 01 | 10 |
| 42 | 3 | 1FFFFF | 7FFFCFF | 1FFFFFFF | 7FFFFFFFF | 1 | 1 | 1 | 1 | 0 | 10 | 10 | 01 |
| 43 | 4 | 1FFFFF | 7FFF9FF | 1FFFFFFF | 7FFFFFFFF | 1 | 1 | 1 | 1 | 0 | 00 | 10 | 10 |
| 44 | 5 | 1FFFFF | 7FFF3FF | 1FFFFFFF | 7FFFFFFFF | 1 | 1 | 1 | 1 | 0 | 11 | 00 | 10 |
| 45 | 6 | 1FFFFF | 7FFE7FE | 1FFFFFFF | 7FFFFFFFF | 1 | 1 | 1 | 1 | 1 | 00 | 11 | 00 |
| 46 | 7 | 1FFFFF | 7FFCFFC | 1FFFFFFF | 7FFFFFFFF | 1 | 1 | 1 | 1 | 0 | 00 | 00 | 11 |
| 47 | 8 | 1FFFFF | 7FF9FF9 | 1FFFFFFF | 7FFFFFFFF | 1 | 1 | 1 | 1 | 0 | 10 | 00 | 00 |
| 48 | 9 | 1FFFFF | 7FF3FF3 | 1FFFFFFF | 7FFFFFFFF | 1 | 1 | 1 | 1 | 0 | 01 | 10 | 00 |
| 49 | 10 | 1FFFFF | 7FE7FE6 | 1FFFFFFF | 7FFFFFFFF | 1 | 1 | 1 | 1 | 1 | 10 | 01 | 10 |
| 50 | 11 | 1FFFFE | 7FFCFFCC | 1FFFFFFE | 7FFFFFFFF | 1 | 1 | 1 | 1 | 0 | 10 | 10 | 01 |
| 51 | 12 | 1FFFFC | 7FF9FF99 | 1FFFFFFC | 7FFFFFFFF | 1 | 1 | 1 | 1 | 0 | 00 | 10 | 10 |
| 52 | 13 | 1FFFF8 | 7FF3FF33 | 1FFFFFF8 | 7FFFFFFFF | 1 | 1 | 1 | 1 | 0 | 11 | 00 | 10 |
| 53 | 14 | 1FFFF0 | 7FE7FE67 | 1FFFFFF0 | 7FFFFFFFF | 1 | 1 | 1 | 1 | 1 | 00 | 11 | 00 |
| 54 | 15 | 1FFFE0 | 7FCFFCCF | 1FFFFFE1 | 7FFFFFFFF | 1 | 1 | 1 | 1 | 0 | 00 | 00 | 11 |
| 55 | 16 | 1FFFC0 | 7F9FF99F | 1FFFFFC3 | 7FFFFFFFF | 1 | 1 | 1 | 1 | 0 | 10 | 00 | 00 |
| 56 | 17 | 1FFF80 | 7F3FF33E | 1FFFFF87 | 7FFFFFFFE | 1 | 0 | 1 | 1 | 1 | 00 | 10 | 00 |
| 57 | 18 | 1FFF00 | 7E7FE67C | 1FFFFF0F | 7FFFFFFFC | 1 | 0 | 1 | 1 | 1 | 00 | 00 | 10 |
| 58 | 19 | 1FFE01 | 7CFFCCF8 | 1FFFFE1E | 7FFFFFFF8 | 1 | 1 | 1 | 1 | 0 | 10 | 00 | 00 |
| 59 | 20 | 1FFC03 | 79FF99F0 | 1FFFFC3C | 7FFFFFFF0 | 1 | 1 | 1 | 1 | 0 | 01 | 10 | 00 |
| 60 | 21 | 1FFF807 | 73FF33E0 | 1FFFF878 | 7FFFFFFE1 | 1 | 1 | 1 | 1 | 1 | 10 | 01 | 10 |
| 61 | 22 | 1FFF00F | 67FE67C0 | 1FFFF0F0 | 7FFFFFFC3 | 1 | 1 | 1 | 1 | 0 | 10 | 10 | 01 |
| 62 | 23 | 1FFE01E | 4FFCCF80 | 1FFFE1E1 | 7FFFFFF87 | 1 | 1 | 1 | 1 | 0 | 00 | 10 | 10 |
| 63 | 24 | 1FFC03C | 1FF99F00 | 1FFFC3C3 | 7FFFFFF0F | 1 | 1 | 1 | 1 | 0 | 11 | 00 | 10 |
| 64 | 25 | 1FF8078 | 3FF33E01 | 1FFF8787 | 7FFFFFE1E | 1 | 1 | 1 | 1 | 1 | 00 | 11 | 00 |
| 65 | 26 | 1FF00F0 | 7FE67C02 | 1FFF0F0F | 7FFFFFC3C | 1 | 1 | 1 | 1 | 0 | 00 | 00 | 11 |
| 66 | 27 | 1FE01E1 | 7FCCF805 | 1FFE1E1E | 7FFFFF878 | 1 | 1 | 1 | 1 | 0 | 10 | 00 | 00 |
| 67 | 28 | 1FC03C3 | 7F99F00A | 1FFC3C3C | 7FFFFF0F0 | 1 | 1 | 1 | 1 | 0 | 01 | 10 | 00 |
| 68 | 29 | 1F80787 | 7F33E015 | 1FFF87878 | 7FFFFE1E1 | 1 | 0 | 1 | 1 | 0 | 10 | 01 | 10 |
| 69 | 30 | 1F00F0F | 7E67C02A | 1FFF0F0F0 | 7FFFFC3C3 | 1 | 0 | 1 | 1 | 1 | 11 | 10 | 01 |
| 70 | 31 | 1E01E1E | 7CCF8054 | 1FFE1E1E1 | 7FFFF8787 | 1 | 1 | 1 | 1 | 1 | 01 | 11 | 10 |
| 71 | 32 | 1C03C3C | 799F00A9 | 1FFC3C3C3 | 7FFFF0F0F | 1 | 1 | 1 | 1 | 1 | 01 | 01 | 11 |
| 72 | 33 | 1807878 | 733E0152 | 1FF878787 | 7FFFE1E1E | 1 | 0 | 1 | 1 | 0 | 00 | 01 | 01 |
| 73 | 34 | 100F0F0 | 667C02A5 | 1FF0F0F0F | 7FFFC3C3C | 0 | 0 | 1 | 1 | 0 | 10 | 00 | 01 |
| 74 | 35 | 001E1E0 | 4CF8054B | 1FE1E1E1F | 7FFF87878 | 0 | 1 | 1 | 1 | 1 | 00 | 10 | 00 |
| 75 | 36 | 003C3C1 | 19F00A96 | 1FC3C3C3F | 7FFF0F0F0 | 0 | 1 | 1 | 1 | 1 | 00 | 00 | 10 |
| 76 | 37 | 0078783 | 33E0152C | 1F878787F | 7FFE1E1E1 | 0 | 1 | 1 | 1 | 1 | 01 | 00 | 00 |
| 77 | 38 | 00F0F07 | 67C02A59 | 1F0F0F0FF | 7FFC3C3C3 | 0 | 1 | 1 | 1 | 0 | 11 | 01 | 00 |
| 78 | 39 | 01E1E0E | 4F8054B3 | 1E1E1E1FF | 7FFF878787 | 0 | 1 | 1 | 1 | 0 | 11 | 11 | 01 |
| 79 | 40 | 03C3C1C | 1F00A966 | 1C3C3C3FF | 7FFF0F0F0F | 0 | 0 | 1 | 1 | 1 | 11 | 11 | 11 |
| 80 | 41 | 0787838 | 3E0152CC | 1878787FF | 7FFE1E1E1E | 0 | 0 | 1 | 1 | 1 | 11 | 11 | 11 |
| 81 | 42 | 0F0F070 | 7C02A598 | 10F0F0FFF | 7FFC3C3C3C | 1 | 0 | 0 | 1 | 1 | 11 | 11 | 11 |
| 82 | 43 | 1E1E0E0 | 78054B30 | 01E1E1FFF | 7FF8787878 | 1 | 0 | 0 | 1 | 1 | 11 | 11 | 11 |
| 83 | 44 | 1C3C1C0 | 700A9660 | 03C3C3FFE | 7FF0F0F0F0 | 1 | 0 | 0 | 1 | 1 | 11 | 11 | 11 |
| 84 | 45 | 1878380 | 60152CC0 | 078787FFC | 7FE1E1E1E0 | 1 | 0 | 0 | 1 | 1 | 11 | 11 | 11 |
| 85 | 46 | 10F0700 | 402A5980 | 0F0F0FFF8 | 7FC3C3C3C0 | 0 | 0 | 1 | 1 | 1 | 11 | 11 | 11 |
| 86 | 47 | 01E0E00 | 0054B300 | 1E1E1FFF0 | 7F87878780 | 0 | 0 | 1 | 1 | 1 | 11 | 11 | 11 |
| 87 | 48 | 03C1C00 | 00A96601 | 1C3C3FFE0 | 7F0F0F0F00 | 0 | 1 | 1 | 0 | 1 | 11 | 11 | 11 |
| 88 | 49 | 0783800 | 0152CC03 | 18787FFC0 | 7E1E1E1E01 | 0 | 0 | 1 | 0 | 0 | 11 | 11 | 11 |

*Sample Data*

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 89 | 50 | 0F07000 | 02A59806 | 10F0FFF80 | 7C3C3C3C03 | 1 | 1 | 0 | 0 | 1 | 11 | 11 | 11 |
| 90 | 51 | 1E0E000 | 054B300D | 01E1FFF00 | 7878787807 | 1 | 0 | 0 | 0 | 0 | 11 | 11 | 11 |
| 91 | 52 | 1C1C001 | 0A96601A | 03C3FFE01 | 70F0F0F00F | 1 | 1 | 0 | 1 | 0 | 10 | 11 | 11 |
| 92 | 53 | 1838003 | 152CC035 | 0787FFC03 | 61E1E1E01E | 1 | 0 | 0 | 1 | 0 | 10 | 10 | 11 |
| 93 | 54 | 1070007 | 2A59806B | 0F0FFF807 | 43C3C3C03C | 0 | 0 | 1 | 1 | 0 | 01 | 10 | 10 |
| 94 | 55 | 00E000F | 54B300D7 | 1E1FFF00F | 0787878078 | 0 | 1 | 1 | 1 | 0 | 10 | 01 | 10 |
| 95 | 56 | 01C001F | 296601AE | 1C3FFE01F | 0F0F0F00F1 | 0 | 0 | 1 | 0 | 1 | 00 | 10 | 01 |
| 96 | 57 | 038003F | 52CC035C | 187FFC03F | 1E1E1E01E2 | 0 | 1 | 1 | 0 | 0 | 00 | 00 | 10 |
| 97 | 58 | 070007F | 259806B8 | 10FFF807F | 3C3C3C03C4 | 0 | 1 | 0 | 0 | 1 | 00 | 00 | 00 |
| 98 | 59 | 0E000FE | 4B300D71 | 01FFF00FE | 7878780788 | 1 | 0 | 0 | 0 | 1 | 00 | 00 | 00 |
| 99 | 60 | 1C001FD | 16601AE2 | 03FFE01FD | 70F0F00F10 | 1 | 0 | 0 | 1 | 0 | 01 | 00 | 00 |
| 100 | 61 | 18003FA | 2CC035C5 | 07FFC03FB | 61E1E01E21 | 1 | 1 | 0 | 1 | 0 | 11 | 01 | 00 |
| 101 | 62 | 10007F4 | 59806B8B | 0FFF807F7 | 43C3C03C43 | 0 | 1 | 1 | 1 | 0 | 11 | 11 | 01 |
| 102 | 63 | 0000FE8 | 3300D717 | 1FFF00FEE | 0787807887 | 0 | 0 | 1 | 1 | 1 | 11 | 11 | 11 |
| 103 | 64 | 0001FD0 | 6601AE2F | 1FFE01FDC | 0F0F00F10E | 0 | 0 | 1 | 0 | 0 | 11 | 11 | 11 |
| 104 | 65 | 0003FA0 | 4C035C5F | 1FFC03FB8 | 1E1E01E21D | 0 | 0 | 1 | 0 | 0 | 11 | 11 | 11 |
| 105 | 66 | 0007F40 | 1806B8BE | 1FF807F70 | 3C3C03C43B | 0 | 0 | 1 | 0 | 0 | 11 | 11 | 11 |
| 106 | 67 | 000FE81 | 300D717C | 1FF00FEE1 | 7878078877 | 0 | 0 | 1 | 0 | 0 | 11 | 11 | 11 |
| 107 | 68 | 001FD02 | 601AE2F8 | 1FE01FDC2 | 70F00F10EF | 0 | 0 | 1 | 1 | 1 | 11 | 11 | 11 |
| 108 | 69 | 003FA05 | 4035C5F0 | 1FC03FB84 | 61E01E21DE | 0 | 0 | 1 | 1 | 1 | 11 | 11 | 11 |
| 109 | 70 | 007F40B | 006B8BE0 | 1F807F708 | 43C03C43BC | 0 | 0 | 1 | 1 | 1 | 11 | 11 | 11 |
| 110 | 71 | 00FE816 | 00D717C0 | 1F00FEE11 | 0780788778 | 0 | 1 | 1 | 1 | 0 | 10 | 11 | 11 |
| 111 | 72 | 01FD02C | 01AE2F81 | 1E01FDC23 | 0F00F10EF1 | 0 | 1 | 1 | 0 | 0 | 10 | 10 | 11 |
| 112 | 73 | 03FA059 | 035C5F02 | 1C03FB847 | 1E01E21DE3 | 0 | 0 | 1 | 0 | 1 | 10 | 10 | 10 |
| 113 | 74 | 07F40B3 | 06B8BE05 | 1807F708F | 3C03C43BC7 | 0 | 1 | 1 | 0 | 0 | 01 | 10 | 10 |
| 114 | 75 | 0FE8166 | 0D717C0B | 100FEE11E | 780788778F | 1 | 0 | 0 | 0 | 0 | 01 | 01 | 10 |
| 115 | 76 | 1FD02CD | 1AE2F817 | 001FDC23D | 700F10EF1F | 1 | 1 | 0 | 0 | 1 | 11 | 01 | 01 |
| 116 | 77 | 1FA059B | 35C5F02F | 003FB847A | 601E21DE3F | 1 | 1 | 0 | 0 | 1 | 10 | 11 | 01 |
| 117 | 78 | 1F40B37 | 6B8BE05E | 007F708F4 | 403C43BC7F | 1 | 1 | 0 | 0 | 0 | 10 | 10 | 11 |
| 118 | 79 | 1E8166E | 5717C0BD | 00FEE11E9 | 00788778FF | 1 | 0 | 0 | 0 | 1 | 10 | 10 | 10 |
| 119 | 80 | 1D02CDC | 2E2F817A | 01FDC23D3 | 00F10EF1FE | 1 | 0 | 0 | 1 | 0 | 01 | 10 | 10 |
| 120 | 81 | 1A059B9 | 5C5F02F5 | 03FB847A6 | 01E21DE3FD | 1 | 0 | 0 | 1 | 1 | 01 | 01 | 10 |
| 121 | 82 | 140B373 | 38BE05EB | 07F708F4C | 03C43BC7FB | 0 | 1 | 0 | 1 | 1 | 11 | 01 | 01 |
| 122 | 83 | 08166E7 | 717C0BD7 | 0FEE11E98 | 0788778FF7 | 1 | 0 | 1 | 1 | 0 | 11 | 11 | 01 |
| 123 | 84 | 102CDCF | 62F817AE | 1FDC23D31 | 0F10EF1FEF | 0 | 1 | 1 | 0 | 1 | 11 | 11 | 11 |
| 124 | 85 | 0059B9F | 45F02F5C | 1FB847A63 | 1E21DE3FDE | 0 | 1 | 1 | 0 | 1 | 11 | 11 | 11 |
| 125 | 86 | 00B373E | 0BE05EB9 | 1F708F4C7 | 3C43BC7FBC | 0 | 1 | 1 | 0 | 1 | 11 | 11 | 11 |
| 126 | 87 | 0166E7D | 17C0BD72 | 1EE11E98F | 788778FF78 | 0 | 1 | 1 | 1 | 0 | 10 | 11 | 11 |
| 127 | 88 | 02CDCFB | 2F817AE5 | 1DC23D31F | 710EF1FEF1 | 0 | 1 | 1 | 0 | 0 | 10 | 10 | 11 |
| 128 | 89 | 059B9F7 | 5F02F5CA | 1B847A63F | 621DE3FDE2 | 0 | 0 | 1 | 0 | 1 | 10 | 10 | 10 |
| 129 | 90 | 0B373EF | 3E05EB94 | 1708F4C7F | 443BC7FBC4 | 1 | 0 | 0 | 0 | 1 | 10 | 10 | 10 |
| 130 | 91 | 166E7DF | 7C0BD728 | 0E11E98FF | 08778FF788 | 0 | 0 | 1 | 0 | 1 | 10 | 10 | 10 |
| 131 | 92 | 0CDCFBE | 7817AE50 | 1C23D31FF | 10EF1FEF10 | 1 | 0 | 1 | 1 | 1 | 01 | 10 | 10 |
| 132 | 93 | 19B9F7D | 702F5CA1 | 1847A63FE | 21DE3FDE21 | 1 | 0 | 1 | 1 | 0 | 10 | 01 | 10 |
| 133 | 94 | 1373EFB | 605EB942 | 108F4C7FC | 43BC7FBC43 | 0 | 0 | 0 | 1 | 1 | 00 | 10 | 01 |
| 134 | 95 | 06E7DF7 | 40BD7285 | 011E98FF8 | 0778FF7886 | 0 | 1 | 0 | 0 | 1 | 01 | 00 | 10 |
| 135 | 96 | 0DCFBEF | 017AE50A | 023D31FF0 | 0EF1FEF10D | 1 | 0 | 0 | 1 | 1 | 00 | 01 | 00 |
| 136 | 97 | 1B9F7DF | 02F5CA15 | 047A63FE1 | 1DE3FDE21A | 1 | 1 | 0 | 1 | 1 | 10 | 00 | 01 |
| 137 | 98 | 173EFBF | 05EB942B | 08F4C7FC3 | 3BC7FBC434 | 0 | 1 | 1 | 1 | 1 | 00 | 10 | 00 |
| 138 | 99 | 0E7DF7F | 0BD72856 | 11E98FF87 | 778FF78869 | 1 | 1 | 0 | 1 | 1 | 00 | 00 | 10 |
| 139 | 100 | 1CFBEFF | 17AE50AC | 03D31FF0F | 6F1FEF10D3 | 1 | 1 | 0 | 0 | 0 | 01 | 00 | 00 |
| 140 | 101 | 19F7DFE | 2F5CA159 | 07A63FE1E | 5E3FDE21A7 | 1 | 0 | 0 | 0 | 0 | 00 | 01 | 00 |
| 141 | 102 | 13EFBFC | 5EB942B3 | 0F4C7FC3C | 3C7FBC434F | 0 | 1 | 1 | 0 | 0 | 10 | 00 | 01 |
| 142 | 103 | 07DF7F8 | 3D728566 | 1E98FF878 | 78FF78869F | 0 | 0 | 1 | 1 | 0 | 00 | 10 | 00 |
| 143 | 104 | 0FBEFF0 | 7AE50ACD | 1D31FF0F0 | 71FEF10D3E | 1 | 1 | 1 | 1 | 0 | 11 | 00 | 10 |
| 144 | 105 | 1F7DFE1 | 75CA159B | 1A63FE1E1 | 63FDE21A7D | 1 | 1 | 1 | 1 | 1 | 00 | 11 | 00 |
| 145 | 106 | 1EFBFC3 | 6B942B36 | 14C7FC3C3 | 47FBC434FB | 1 | 1 | 0 | 1 | 1 | 11 | 00 | 11 |

*Sample Data*

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 146 | 107 | 1DF7F86 | 5728566D | 098FF8786 | 0FF78869F7 | 1 | 0 | 1 | 1 | 0 | 00 | 11 | 00 |
| 147 | 108 | 1BEFF0C | 2E50ACDB | 131FF0F0C | 1FEF10D3EF | 1 | 0 | 0 | 1 | 0 | 11 | 00 | 11 |
| 148 | 109 | 17DFE19 | 5CA159B6 | 063FE1E19 | 3FDE21A7DF | 0 | 1 | 0 | 1 | 1 | 01 | 11 | 00 |
| 149 | 110 | 0FBFC33 | 3942B36D | 0C7FC3C32 | 7FBC434FBF | 1 | 0 | 1 | 1 | 0 | 01 | 01 | 11 |
| 150 | 111 | 1F7F866 | 728566DB | 18FF87865 | 7F78869F7E | 1 | 1 | 1 | 0 | 0 | 00 | 01 | 01 |
| 151 | 112 | 1EFF0CC | 650ACDB6 | 11FF0F0CB | 7EF10D3EFC | 1 | 0 | 0 | 1 | 0 | 10 | 00 | 01 |
| 152 | 113 | 1DFE199 | 4A159B6D | 03FE1E196 | 7DE21A7DF9 | 1 | 0 | 0 | 1 | 0 | 00 | 10 | 00 |
| 153 | 114 | 1BFC333 | 142B36DB | 07FC3C32C | 7BC434FBF3 | 1 | 0 | 0 | 1 | 0 | 00 | 00 | 10 |
| 154 | 115 | 17F8666 | 28566DB6 | 0FF878659 | 778869F7E6 | 0 | 0 | 1 | 1 | 0 | 01 | 00 | 00 |
| 155 | 116 | 0FF0CCC | 50ACDB6D | 1FF0F0CB3 | 6F10D3EFCC | 1 | 1 | 1 | 0 | 0 | 11 | 01 | 00 |
| 156 | 117 | 1FE1999 | 2159B6DA | 1FE1E1966 | 5E21A7DF99 | 1 | 0 | 1 | 0 | 1 | 10 | 11 | 01 |
| 157 | 118 | 1FC3332 | 42B36DB5 | 1FC3C32CC | 3C434FBF33 | 1 | 1 | 1 | 0 | 1 | 10 | 10 | 11 |
| 158 | 119 | 1F86664 | 0566DB6B | 1F8786599 | 78869F7E67 | 1 | 0 | 1 | 1 | 1 | 01 | 10 | 10 |
| 159 | 120 | 1F0CCC8 | 0ACDB6D6 | 1F0F0CB33 | 710D3EFCCE | 1 | 1 | 1 | 0 | 0 | 10 | 01 | 10 |
| 160 | 121 | 1E19991 | 159B6DAC | 1E1E19666 | 621A7DF99D | 1 | 1 | 1 | 0 | 1 | 11 | 10 | 01 |
| 161 | 122 | 1C33323 | 2B36DB58 | 1C3C32CCC | 4434FBF33B | 1 | 0 | 1 | 0 | 1 | 00 | 11 | 10 |
| 162 | 123 | 1866647 | 566DB6B0 | 187865999 | 0869F7E676 | 1 | 0 | 1 | 0 | 0 | 11 | 00 | 11 |
| 163 | 124 | 10CCC8F | 2CDB6D60 | 10F0CB333 | 10D3EFCCEC | 0 | 1 | 0 | 1 | 1 | 01 | 11 | 00 |
| 164 | 125 | 019991E | 59B6DAC0 | 01E196666 | 21A7DF99D9 | 0 | 1 | 0 | 1 | 1 | 10 | 01 | 11 |
| 165 | 126 | 033323C | 336DB580 | 03C32CCCD | 434FBF33B3 | 0 | 0 | 0 | 0 | 0 | 00 | 10 | 01 |
| 166 | 127 | 0666478 | 66DB6B01 | 07865999A | 069F7E6766 | 0 | 1 | 0 | 1 | 0 | 00 | 00 | 10 |
| 167 | 128 | 0CCC8F0 | 4DB6D603 | 0F0CB3334 | 0D3EFCCECD | 1 | 1 | 1 | 0 | 1 | 01 | 00 | 00 |
| 168 | 129 | 19991E1 | 1B6DAC07 | 1E1966669 | 1A7DF99D9B | 1 | 0 | 1 | 0 | 1 | 00 | 01 | 00 |
| 169 | 130 | 13323C3 | 36DB580E | 1C32CCCD3 | 34FBF33B37 | 0 | 1 | 1 | 1 | 1 | 10 | 00 | 01 |
| 170 | 131 | 0664786 | 6DB6B01C | 1865999A7 | 69F7E6766F | 0 | 1 | 1 | 1 | 1 | 00 | 10 | 00 |
| 171 | 132 | 0CC8F0D | 5B6D6039 | 10CB3334F | 53EFCCECDF | 1 | 0 | 0 | 1 | 0 | 00 | 00 | 10 |
| 172 | 133 | 1991E1A | 36DAC073 | 01966669E | 27DF99D9BF | 1 | 1 | 0 | 1 | 1 | 01 | 00 | 00 |
| 173 | 134 | 1323C35 | 6DB580E6 | 032CCCD3C | 4FBF33B37E | 0 | 1 | 0 | 1 | 1 | 00 | 01 | 00 |
| 174 | 135 | 064786A | 5B6B01CD | 065999A78 | 1F7E6766FC | 0 | 0 | 0 | 0 | 0 | 11 | 00 | 01 |
| 175 | 136 | 0C8F0D5 | 36D6039B | 0CB3334F0 | 3EFCCECDF9 | 1 | 1 | 1 | 1 | 1 | 00 | 11 | 00 |
| 176 | 137 | 191E1AA | 6DAC0737 | 1966669E1 | 7DF99D9BF3 | 1 | 1 | 1 | 1 | 0 | 00 | 00 | 11 |
| 177 | 138 | 123C354 | 5B580E6E | 12CCCD3C3 | 7BF33B37E7 | 0 | 0 | 0 | 1 | 1 | 00 | 00 | 00 |
| 178 | 139 | 04786A9 | 36B01CDC | 05999A787 | 77E6766FCE | 0 | 1 | 0 | 1 | 0 | 01 | 00 | 00 |
| 179 | 140 | 08F0D53 | 6D6039B8 | 0B3334F0E | 6FCCECDF9C | 1 | 0 | 1 | 1 | 0 | 11 | 01 | 00 |
| 180 | 141 | 11E1AA6 | 5AC07370 | 166669E1D | 5F99D9BF38 | 0 | 1 | 0 | 1 | 1 | 10 | 11 | 01 |
| 181 | 142 | 03C354C | 3580E6E0 | 0CCCD3C3A | 3F33B37E70 | 0 | 1 | 1 | 0 | 0 | 10 | 10 | 11 |
| 182 | 143 | 0786A99 | 6B01CDC0 | 1999A7875 | 7E6766FCE1 | 0 | 0 | 1 | 0 | 1 | 10 | 10 | 10 |
| 183 | 144 | 0F0D533 | 56039B81 | 13334F0EB | 7CCECDF9C2 | 1 | 0 | 0 | 1 | 0 | 01 | 10 | 10 |
| 184 | 145 | 1E1AA66 | 2C073703 | 06669E1D6 | 799D9BF385 | 1 | 0 | 0 | 1 | 1 | 01 | 01 | 10 |
| 185 | 146 | 1C354CC | 580E6E06 | 0CCD3C3AC | 733B37E70B | 1 | 0 | 1 | 0 | 1 | 11 | 01 | 01 |
| 186 | 147 | 186A998 | 301CDC0C | 199A78759 | 66766FCE17 | 1 | 0 | 1 | 0 | 1 | 10 | 11 | 01 |
| 187 | 148 | 10D5331 | 6039B818 | 1334F0EB2 | 4CECDF9C2F | 0 | 0 | 0 | 1 | 1 | 01 | 10 | 11 |
| 188 | 149 | 01AA662 | 40737031 | 0669E1D65 | 19D9BF385E | 0 | 0 | 0 | 1 | 0 | 01 | 01 | 10 |
| 189 | 150 | 0354CC5 | 00E6E063 | 0CD3C3ACB | 33B37E70BD | 0 | 1 | 1 | 1 | 0 | 00 | 01 | 01 |
| 190 | 151 | 06A998A | 01CDC0C6 | 19A787596 | 6766FCE17B | 0 | 1 | 1 | 0 | 0 | 10 | 00 | 01 |
| 191 | 152 | 0D53315 | 039B818C | 134F0EB2C | 4ECDF9C2F6 | 1 | 1 | 0 | 1 | 1 | 00 | 10 | 00 |
| 192 | 153 | 1AA662A | 07370318 | 069E1D659 | 1D9BF385ED | 1 | 0 | 0 | 1 | 0 | 00 | 00 | 10 |
| 193 | 154 | 154CC54 | 0E6E0630 | 0D3C3ACB3 | 3B37E70BDB | 0 | 0 | 1 | 0 | 1 | 00 | 00 | 00 |
| 194 | 155 | 0A998A8 | 1CDC0C60 | 1A7875967 | 766FCE17B6 | 1 | 1 | 1 | 0 | 1 | 01 | 00 | 00 |
| 195 | 156 | 1533151 | 39B818C0 | 14F0EB2CE | 6CDF9C2F6C | 0 | 1 | 0 | 1 | 1 | 00 | 01 | 00 |
| 196 | 157 | 0A662A3 | 73703180 | 09E1D659D | 59BF385ED8 | 1 | 0 | 1 | 1 | 1 | 10 | 00 | 01 |
| 197 | 158 | 14CC547 | 66E06301 | 13C3ACB3A | 337E70BDB0 | 0 | 1 | 0 | 0 | 1 | 11 | 10 | 00 |
| 198 | 159 | 0998A8E | 4DC0C602 | 078759675 | 66FCE17B61 | 1 | 1 | 0 | 1 | 0 | 01 | 11 | 10 |
| 199 | 160 | 133151D | 1B818C05 | 0F0EB2CEB | 4DF9C2F6C2 | 0 | 1 | 1 | 1 | 0 | 01 | 01 | 11 |
| 200 | 161 | 0662A3B | 3703180B | 1E1D659D6 | 1BF385ED85 | 0 | 0 | 1 | 1 | 1 | 11 | 01 | 01 |
| 201 | 162 | 0CC5477 | 6E063017 | 1C3ACB3AC | 37E70BDB0B | 1 | 0 | 1 | 1 | 0 | 11 | 11 | 01 |
| 202 | 163 | 198A8EF | 5C0C602F | 187596759 | 6FCE17B617 | 1 | 0 | 1 | 1 | 0 | 10 | 11 | 11 |

*Sample Data*

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 203 | 164 | 13151DE | 3818C05F | 10EB2CEB2 | 5F9C2F6C2F | 0 | 0 | 0 | 1 | 1 | 01 | 10 | 11 |
| 204 | 165 | 062A3BC | 703180BF | 01D659D65 | 3F385ED85E | 0 | 0 | 0 | 0 | 1 | 00 | 01 | 10 |
| 205 | 166 | 0C54779 | 6063017E | 03ACB3ACB | 7E70BDB0BD | 1 | 0 | 0 | 0 | 1 | 11 | 00 | 01 |
| 206 | 167 | 18A8EF2 | 40C602FD | 075967597 | 7CE17B617B | 1 | 1 | 0 | 1 | 0 | 00 | 11 | 00 |
| 207 | 168 | 1151DE4 | 018C05FA | 0EB2CEB2F | 79C2F6C2F7 | 0 | 1 | 1 | 1 | 1 | 11 | 00 | 11 |
| 208 | 169 | 02A3BC9 | 03180BF5 | 1D659D65E | 7385ED85EE | 0 | 0 | 1 | 1 | 1 | 01 | 11 | 00 |
| 209 | 170 | 0547793 | 063017EB | 1ACB3ACBC | 670BDB0BDC | 0 | 0 | 1 | 0 | 0 | 10 | 01 | 11 |
| 210 | 171 | 0A8EF27 | 0C602FD6 | 159675978 | 4E17B617B9 | 1 | 0 | 0 | 0 | 1 | 00 | 10 | 01 |
| 211 | 172 | 151DE4E | 18C05FAD | 0B2CEB2F1 | 1C2F6C2F73 | 0 | 1 | 1 | 0 | 0 | 00 | 00 | 10 |
| 212 | 173 | 0A3BC9C | 3180BF5A | 1659D65E3 | 385ED85EE6 | 1 | 1 | 0 | 0 | 0 | 01 | 00 | 00 |
| 213 | 174 | 1477938 | 63017EB5 | 0CB3ACBC6 | 70BDB0BDCC | 0 | 0 | 1 | 1 | 1 | 00 | 01 | 00 |
| 214 | 175 | 08EF270 | 4602FD6A | 19675978D | 617B617B99 | 1 | 0 | 1 | 0 | 0 | 10 | 00 | 01 |
| 215 | 176 | 11DE4E1 | 0C05FAD5 | 12CEB2F1A | 42F6C2F733 | 0 | 0 | 0 | 1 | 1 | 11 | 10 | 00 |
| 216 | 177 | 03BC9C3 | 180BF5AA | 059D65E34 | 05ED85EE67 | 0 | 0 | 0 | 1 | 0 | 00 | 11 | 10 |
| 217 | 178 | 0779387 | 3017EB55 | 0B3ACBC68 | 0BDB0BDCCF | 0 | 0 | 1 | 1 | 0 | 11 | 00 | 11 |
| 218 | 179 | 0EF270F | 602FD6AA | 1675978D0 | 17B617B99F | 1 | 0 | 0 | 1 | 1 | 01 | 11 | 00 |
| 219 | 180 | 1DE4E1F | 405FAD54 | 0CEB2F1A1 | 2F6C2F733F | 1 | 0 | 1 | 0 | 1 | 10 | 01 | 11 |
| 220 | 181 | 1BC9C3F | 00BF5AA9 | 19D65E342 | 5ED85EE67F | 1 | 1 | 1 | 1 | 0 | 10 | 10 | 01 |
| 221 | 182 | 179387F | 017EB552 | 13ACBC684 | 3DB0BDCCFE | 0 | 0 | 0 | 1 | 1 | 10 | 10 | 10 |
| 222 | 183 | 0F270FF | 02FD6AA5 | 075978D09 | 7B617B99FC | 1 | 1 | 0 | 0 | 0 | 01 | 10 | 10 |
| 223 | 184 | 1E4E1FF | 05FAD54A | 0EB2F1A12 | 76C2F733F9 | 1 | 1 | 1 | 1 | 1 | 10 | 01 | 10 |
| 224 | 185 | 1C9C3FE | 0BF5AA94 | 1D65E3425 | 6D85EE67F2 | 1 | 1 | 1 | 1 | 0 | 10 | 10 | 01 |
| 225 | 186 | 19387FD | 17EB5529 | 1ACBC684B | 5B0BDCCFE4 | 1 | 1 | 1 | 0 | 1 | 01 | 10 | 10 |
| 226 | 187 | 1270FFA | 2FD6AA53 | 15978D096 | 3617B99FC9 | 0 | 1 | 0 | 0 | 0 | 01 | 01 | 10 |
| 227 | 188 | 04E1FF5 | 5FAD54A7 | 0B2F1A12C | 6C2F733F93 | 0 | 1 | 1 | 0 | 1 | 11 | 01 | 01 |
| 228 | 189 | 09C3FEB | 3F5AA94E | 165E34258 | 585EE67F27 | 1 | 0 | 0 | 0 | 0 | 10 | 11 | 01 |
| 229 | 190 | 1387FD7 | 7EB5529C | 0CBC684B1 | 30BDCCFE4F | 0 | 1 | 1 | 1 | 1 | 10 | 10 | 11 |
| 230 | 191 | 070FFAE | 7D6AA538 | 1978D0962 | 617B99FC9E | 0 | 0 | 1 | 0 | 1 | 10 | 10 | 10 |
| 231 | 192 | 0E1FF5C | 7AD54A70 | 12F1A12C4 | 42F733F93D | 1 | 1 | 0 | 1 | 1 | 01 | 10 | 10 |
| 232 | 193 | 1C3FEB9 | 75AA94E1 | 05E342588 | 05EE67F27A | 1 | 1 | 0 | 1 | 0 | 10 | 01 | 10 |
| 233 | 194 | 187FD73 | 6B5529C3 | 0BC684B10 | 0BDCCFE4F4 | 1 | 0 | 1 | 1 | 1 | 11 | 10 | 01 |
| 234 | 195 | 10FFAE6 | 56AA5386 | 178D09621 | 17B99FC9E8 | 0 | 1 | 0 | 1 | 1 | 00 | 11 | 10 |
| 235 | 196 | 01FF5CC | 2D54A70C | 0F1A12C43 | 2F733F93D0 | 0 | 0 | 1 | 0 | 1 | 10 | 00 | 11 |
| 236 | 197 | 03FEB98 | 5AA94E19 | 1E3425887 | 5EE67F27A1 | 0 | 1 | 1 | 1 | 1 | 00 | 10 | 00 |
| 237 | 198 | 07FD731 | 35529C33 | 1C684B10F | 3DCCFE4F42 | 0 | 0 | 1 | 1 | 0 | 00 | 00 | 10 |
| 238 | 199 | 0FFAE63 | 6AA53866 | 18D09621F | 7B99FC9E84 | 1 | 1 | 1 | 1 | 0 | 10 | 00 | 00 |
| 239 | 200 | 1FF5CC6 | 554A70CD | 11A12C43F | 7733F93D09 | 1 | 0 | 0 | 0 | 1 | 11 | 10 | 00 |

```
Z[0]  = 59
Z[1]  = 3B
Z[2]  = EF
Z[3]  = 07
Z[4]  = 13
Z[5]  = 70
Z[6]  = 9B
Z[7]  = B7
Z[8]  = 52
Z[9]  = 8F
Z[10] = 3E
Z[11] = B9
Z[12] = A5
Z[13] = AC
Z[14] = EA
Z[15] = 9E
```

*Sample Data*

```
================================================================================================
         Reload this pattern into the LFSRs
         Hold content of Summation Combiner regs and calculate new C[t+1] and Z values
================================================================================================
LFSR1  <= 1521359
LFSR2  <= 528F703B
LFSR3  <= 0AC3E9BEF
LFSR4  <= 4FEAB9B707
C[t+1] <= 00

================================================================================================
         Generating 125 key symbols (encryption/decryption sequence)
================================================================================================
```

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 240 | 1 | 1521359 | 528F703B | 0AC3E9BEF | 4FEAB9B707 | 0 | 1 | 1 | 1 | 1 | 00 | 10 | 00 |
| 241 | 2 | 0A426B3 | 251EE076 | 1587D37DE | 1FD5736E0F | 1 | 0 | 0 | 1 | 0 | 00 | 00 | 10 |
| 242 | 3 | 1484D67 | 4A3DC0ED | 0B0FA6FBD | 3FAAE6DC1E | 0 | 0 | 1 | 1 | 0 | 01 | 00 | 00 |
| 243 | 4 | 0909ACF | 147B81DA | 161F4DF7A | 7F55CDB83D | 1 | 0 | 0 | 0 | 0 | 00 | 01 | 00 |
| 244 | 5 | 121359E | 28F703B5 | 0C3E9BEF5 | 7EAB9B707B | 0 | 1 | 1 | 1 | 1 | 10 | 00 | 01 |
| 245 | 6 | 0426B3C | 51EE076B | 187D37DEB | 7D5736E0F6 | 0 | 1 | 1 | 0 | 0 | 00 | 10 | 00 |
| 246 | 7 | 084D679 | 23DC0ED6 | 10FA6FBD7 | 7AAE6DC1EC | 1 | 1 | 0 | 1 | 1 | 00 | 00 | 10 |
| 247 | 8 | 109ACF2 | 47B81DAC | 01F4DF7AF | 755CDB83D8 | 0 | 1 | 0 | 0 | 1 | 00 | 00 | 00 |
| 248 | 9 | 01359E4 | 0F703B59 | 03E9BEF5E | 6AB9B707B1 | 0 | 0 | 0 | 1 | 1 | 00 | 00 | 00 |
| 249 | 10 | 026B3C8 | 1EE076B3 | 07D37DEBD | 55736E0F63 | 0 | 1 | 0 | 0 | 1 | 00 | 00 | 00 |
| 250 | 11 | 04D6791 | 3DC0ED67 | 0FA6FBD7A | 2AE6DC1EC7 | 0 | 1 | 1 | 1 | 1 | 01 | 00 | 00 |
| 251 | 12 | 09ACF22 | 7B81DACF | 1F4DF7AF4 | 55CDB83D8F | 1 | 1 | 1 | 1 | 1 | 11 | 01 | 00 |
| 252 | 13 | 1359E44 | 7703B59E | 1E9BEF5E8 | 2B9B707B1F | 0 | 0 | 1 | 1 | 1 | 10 | 11 | 01 |
| 253 | 14 | 06B3C88 | 6E076B3C | 1D37DEBD0 | 5736E0F63F | 0 | 0 | 1 | 0 | 1 | 01 | 10 | 11 |
| 254 | 15 | 0D67911 | 5C0ED678 | 1A6FBD7A1 | 2E6DC1EC7E | 1 | 0 | 1 | 0 | 1 | 01 | 01 | 10 |
| 255 | 16 | 1ACF223 | 381DACF0 | 14DF7AF42 | 5CDB83D8FD | 1 | 0 | 0 | 1 | 1 | 11 | 01 | 01 |
| 256 | 17 | 159E446 | 703B59E0 | 09BEF5E85 | 39B707B1FA | 0 | 0 | 1 | 1 | 1 | 10 | 11 | 01 |
| 257 | 18 | 0B3C88C | 6076B3C0 | 137DEBD0A | 736E0F63F4 | 1 | 0 | 0 | 0 | 1 | 01 | 10 | 11 |
| 258 | 19 | 1679118 | 40ED6780 | 06FBD7A15 | 66DC1EC7E8 | 0 | 1 | 0 | 1 | 1 | 01 | 01 | 10 |
| 259 | 20 | 0CF2231 | 01DACF00 | 0DF7AF42A | 4DB83D8FD1 | 1 | 1 | 1 | 1 | 1 | 00 | 01 | 01 |
| 260 | 21 | 19E4463 | 03B59E01 | 1BEF5E854 | 1B707B1FA3 | 1 | 1 | 1 | 0 | 1 | 10 | 00 | 01 |
| 261 | 22 | 13C88C6 | 076B3C03 | 17DEBD0A9 | 36E0F63F47 | 0 | 0 | 0 | 1 | 1 | 11 | 10 | 00 |
| 262 | 23 | 079118C | 0ED67807 | 0FBD7A152 | 6DC1EC7E8E | 0 | 1 | 1 | 1 | 0 | 01 | 11 | 10 |
| 263 | 24 | 0F22318 | 1DACF00E | 1F7AF42A4 | 5B83D8FD1D | 1 | 1 | 1 | 1 | 1 | 01 | 01 | 11 |
| 264 | 25 | 1E44630 | 3B59E01C | 1EF5E8548 | 3707B1FA3B | 1 | 0 | 1 | 0 | 1 | 11 | 01 | 01 |
| 265 | 26 | 1C88C61 | 76B3C039 | 1DEBD0A91 | 6E0F63F477 | 1 | 1 | 1 | 0 | 0 | 11 | 11 | 01 |
| 266 | 27 | 19118C3 | 6D678073 | 1BD7A1523 | 5C1EC7E8EF | 1 | 0 | 1 | 0 | 1 | 11 | 11 | 11 |
| 267 | 28 | 1223187 | 5ACF00E6 | 17AF42A46 | 383D8FD1DE | 0 | 1 | 0 | 0 | 0 | 11 | 11 | 11 |
| 268 | 29 | 044630E | 359E01CC | 0F5E8548D | 707B1FA3BD | 0 | 1 | 1 | 0 | 1 | 11 | 11 | 11 |
| 269 | 30 | 088C61C | 6B3C0399 | 1EBD0A91A | 60F63F477B | 1 | 0 | 1 | 1 | 0 | 10 | 11 | 11 |
| 270 | 31 | 1118C39 | 56780733 | 1D7A15234 | 41EC7E8EF6 | 0 | 0 | 1 | 1 | 0 | 10 | 10 | 11 |
| 271 | 32 | 0231872 | 2CF00E67 | 1AF42A468 | 03D8FD1DEC | 0 | 1 | 1 | 1 | 1 | 01 | 10 | 10 |
| 272 | 33 | 04630E5 | 59E01CCE | 15E8548D1 | 07B1FA3BD8 | 0 | 1 | 0 | 1 | 1 | 01 | 01 | 10 |
| 273 | 34 | 08C61CB | 33C0399D | 0BD0A91A3 | 0F63F477B1 | 1 | 1 | 1 | 0 | 0 | 00 | 01 | 01 |
| 274 | 35 | 118C396 | 6780733A | 17A152347 | 1EC7E8EF63 | 0 | 1 | 0 | 1 | 0 | 10 | 00 | 01 |
| 275 | 36 | 031872D | 4F00E674 | 0F42A468E | 3D8FD1DEC7 | 0 | 0 | 1 | 1 | 0 | 00 | 10 | 00 |
| 276 | 37 | 0630E5A | 1E01CCE8 | 1E8548D1D | 7B1FA3BD8E | 0 | 0 | 1 | 0 | 1 | 01 | 00 | 10 |
| 277 | 38 | 0C61CB5 | 3C0399D0 | 1D0A91A3B | 763F477B1C | 1 | 0 | 1 | 0 | 1 | 00 | 01 | 00 |
| 278 | 39 | 18C396A | 780733A0 | 1A1523477 | 6C7E8EF639 | 1 | 0 | 1 | 0 | 0 | 10 | 00 | 01 |
| 279 | 40 | 11872D5 | 700E6741 | 142A468EF | 58FD1DEC72 | 0 | 0 | 0 | 1 | 1 | 11 | 10 | 00 |
| 280 | 41 | 030E5AB | 601CCE83 | 08548D1DF | 31FA3BD8E5 | 0 | 0 | 1 | 1 | 1 | 00 | 11 | 10 |
| 281 | 42 | 061CB57 | 40399D07 | 10A91A3BF | 63F477B1CB | 0 | 0 | 0 | 1 | 1 | 10 | 00 | 11 |
| 282 | 43 | 0C396AF | 00733A0F | 01523477E | 47E8EF6396 | 1 | 0 | 0 | 1 | 0 | 00 | 10 | 00 |
| 283 | 44 | 1872D5F | 00E6741F | 02A468EFD | 0FD1DEC72C | 1 | 1 | 0 | 1 | 1 | 00 | 00 | 10 |

*Sample Data*

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 284 | 45 | 10E5ABE | 01CCE83F | 0548D1DFA | 1FA3BD8E58 | 0 | 1 | 0 | 1 | 0 | 01 | 00 | 00 |
| 285 | 46 | 01CB57C | 0399D07F | 0A91A3BF4 | 3F477B1CB0 | 0 | 1 | 1 | 0 | 1 | 00 | 01 | 00 |
| 286 | 47 | 0396AF9 | 0733A0FE | 1523477E9 | 7E8EF63961 | 0 | 0 | 0 | 1 | 1 | 11 | 00 | 01 |
| 287 | 48 | 072D5F3 | 0E6741FD | 0A468EFD2 | 7D1DEC72C3 | 0 | 0 | 1 | 0 | 0 | 01 | 11 | 00 |
| 288 | 49 | 0E5ABE7 | 1CCE83FA | 148D1DFA4 | 7A3BD8E587 | 1 | 1 | 0 | 0 | 1 | 10 | 01 | 11 |
| 289 | 50 | 1CB57CE | 399D07F4 | 091A3BF49 | 7477B1CB0F | 1 | 1 | 1 | 0 | 1 | 11 | 10 | 01 |
| 290 | 51 | 196AF9D | 733A0FE9 | 123477E92 | 68EF63961E | 1 | 0 | 0 | 1 | 1 | 00 | 11 | 10 |
| 291 | 52 | 12D5F3B | 66741FD2 | 0468EFD25 | 51DEC72C3C | 0 | 0 | 0 | 1 | 1 | 10 | 00 | 11 |
| 292 | 53 | 05ABE77 | 4CE83FA4 | 08D1DFA4B | 23BD8E5879 | 0 | 1 | 1 | 1 | 1 | 00 | 10 | 00 |
| 293 | 54 | 0B57CEE | 19D07F49 | 11A3BF496 | 477B1CB0F2 | 1 | 1 | 0 | 0 | 0 | 00 | 00 | 10 |
| 294 | 55 | 16AF9DC | 33A0FE92 | 03477E92C | 0EF63961E4 | 0 | 1 | 0 | 1 | 0 | 01 | 00 | 00 |
| 295 | 56 | 0D5F3B8 | 6741FD25 | 068EFD259 | 1DEC72C3C9 | 1 | 0 | 0 | 1 | 1 | 00 | 01 | 00 |
| 296 | 57 | 1ABE771 | 4E83FA4B | 0D1DFA4B3 | 3BD8E58793 | 1 | 1 | 1 | 1 | 0 | 01 | 00 | 01 |
| 297 | 58 | 157CEE2 | 1D07F496 | 1A3BF4967 | 77B1CB0F26 | 0 | 0 | 1 | 1 | 1 | 00 | 01 | 00 |
| 298 | 59 | 0AF9DC5 | 3A0FE92D | 1477E92CE | 6F63961E4D | 1 | 0 | 0 | 0 | 1 | 11 | 00 | 01 |
| 299 | 60 | 15F3B8B | 741FD25A | 08EFD259C | 5EC72C3C9B | 0 | 0 | 1 | 1 | 1 | 01 | 11 | 00 |
| 300 | 61 | 0BE7716 | 683FA4B4 | 11DFA4B39 | 3D8E587937 | 1 | 0 | 0 | 1 | 1 | 10 | 01 | 11 |
| 301 | 62 | 17CEE2D | 507F4968 | 03BF49672 | 7B1CB0F26E | 0 | 0 | 0 | 0 | 0 | 00 | 10 | 01 |
| 302 | 63 | 0F9DC5B | 20FE92D0 | 077E92CE4 | 763961E4DC | 1 | 1 | 0 | 0 | 0 | 00 | 00 | 10 |
| 303 | 64 | 1F3B8B6 | 41FD25A0 | 0EFD259C9 | 6C72C3C9B9 | 1 | 1 | 1 | 0 | 1 | 01 | 00 | 00 |
| 304 | 65 | 1E7716D | 03FA4B40 | 1DFA4B393 | 58E5879373 | 1 | 1 | 1 | 1 | 1 | 11 | 01 | 00 |
| 305 | 66 | 1CEE2DB | 07F49680 | 1BF496727 | 31CB0F26E6 | 1 | 1 | 1 | 1 | 1 | 11 | 11 | 01 |
| 306 | 67 | 19DC5B7 | 0FE92D00 | 17E92CE4E | 63961E4DCD | 1 | 1 | 0 | 1 | 0 | 10 | 11 | 11 |
| 307 | 68 | 13B8B6F | 1FD25A00 | 0FD259C9C | 472C3C9B9A | 0 | 1 | 1 | 0 | 0 | 10 | 10 | 11 |
| 308 | 69 | 07716DF | 3FA4B400 | 1FA4B3938 | 0E58793735 | 0 | 1 | 1 | 0 | 0 | 01 | 10 | 10 |
| 309 | 70 | 0EE2DBF | 7F496800 | 1F4967271 | 1CB0F26E6A | 1 | 0 | 1 | 1 | 0 | 10 | 01 | 10 |
| 310 | 71 | 1DC5B7F | 7E92D000 | 1E92CE4E2 | 3961E4DCD4 | 1 | 1 | 1 | 0 | 1 | 11 | 10 | 01 |
| 311 | 72 | 1B8B6FF | 7D25A001 | 1D259C9C4 | 72C3C9B9A9 | 1 | 0 | 1 | 1 | 0 | 01 | 11 | 10 |
| 312 | 73 | 1716DFF | 7A4B4002 | 1A4B39389 | 6587937352 | 0 | 0 | 1 | 1 | 1 | 10 | 01 | 11 |
| 313 | 74 | 0E2DBFF | 74968005 | 149672713 | 4B0F26E6A5 | 1 | 1 | 0 | 0 | 0 | 11 | 10 | 01 |
| 314 | 75 | 1C5B7FE | 692D000B | 092CE4E26 | 161E4DCD4B | 1 | 0 | 1 | 0 | 1 | 00 | 11 | 10 |
| 315 | 76 | 18B6FFC | 525A0017 | 1259C9C4D | 2C3C9B9A96 | 1 | 0 | 0 | 0 | 1 | 10 | 00 | 11 |
| 316 | 77 | 116DFF8 | 24B4002F | 04B39389B | 587937352C | 0 | 1 | 0 | 0 | 1 | 11 | 10 | 00 |
| 317 | 78 | 02DBFF1 | 4968005F | 096727136 | 30F26E6A58 | 0 | 0 | 1 | 1 | 1 | 00 | 11 | 10 |
| 318 | 79 | 05B7FE3 | 12D000BF | 12CE4E26C | 61E4DCD4B1 | 0 | 1 | 0 | 1 | 0 | 11 | 00 | 11 |
| 319 | 80 | 0B6FFC7 | 25A0017F | 059C9C4D8 | 43C9B9A963 | 1 | 1 | 0 | 1 | 0 | 00 | 11 | 00 |
| 320 | 81 | 16DFF8E | 4B4002FF | 0B39389B1 | 07937352C6 | 0 | 0 | 1 | 1 | 0 | 11 | 00 | 11 |
| 321 | 82 | 0DBFF1C | 168005FF | 167271363 | 0F26E6A58C | 1 | 1 | 0 | 0 | 1 | 01 | 11 | 00 |
| 322 | 83 | 1B7FE38 | 2D000BFF | 0CE4E26C7 | 1E4DCD4B18 | 1 | 0 | 1 | 0 | 1 | 10 | 01 | 11 |
| 323 | 84 | 16FFC70 | 5A0017FF | 19C9C4D8F | 3C9B9A9631 | 0 | 0 | 1 | 1 | 0 | 11 | 10 | 01 |
| 324 | 85 | 0DFF8E1 | 34002FFF | 139389B1E | 7937352C62 | 1 | 0 | 0 | 0 | 0 | 00 | 11 | 10 |
| 325 | 86 | 1BFF1C3 | 68005FFF | 07271363D | 726E6A58C4 | 1 | 0 | 0 | 0 | 1 | 10 | 00 | 11 |
| 326 | 87 | 17FE387 | 5000BFFE | 0E4E26C7B | 64DCD4B188 | 0 | 0 | 1 | 1 | 0 | 00 | 10 | 00 |
| 327 | 88 | 0FFC70F | 20017FFD | 1C9C4D8F6 | 49B9A96311 | 1 | 0 | 1 | 1 | 1 | 00 | 00 | 10 |
| 328 | 89 | 1FF8E1F | 4002FFFB | 19389B1ED | 137352C623 | 1 | 0 | 1 | 0 | 0 | 01 | 00 | 00 |
| 329 | 90 | 1FF1C3F | 0005FFF7 | 1271363DB | 26E6A58C46 | 1 | 0 | 0 | 1 | 1 | 00 | 01 | 00 |
| 330 | 91 | 1FE387F | 000BFFEE | 04E26C7B6 | 4DCD4B188C | 1 | 0 | 0 | 1 | 0 | 10 | 00 | 01 |
| 331 | 92 | 1FC70FF | 0017FFDC | 09C4D8F6D | 1B9A963118 | 1 | 0 | 1 | 1 | 1 | 00 | 10 | 00 |
| 332 | 93 | 1F8E1FF | 002FFFB8 | 1389B1EDA | 37352C6231 | 1 | 0 | 0 | 0 | 1 | 01 | 00 | 10 |
| 333 | 94 | 1F1C3FF | 005FFF70 | 071363DB4 | 6E6A58C462 | 1 | 0 | 0 | 0 | 0 | 00 | 01 | 00 |
| 334 | 95 | 1E387FE | 00BFFEE0 | 0E26C7B68 | 5CD4B188C5 | 1 | 1 | 1 | 1 | 0 | 01 | 00 | 01 |
| 335 | 96 | 1C70FFC | 017FFDC1 | 1C4D8F6D1 | 39A963118A | 1 | 0 | 1 | 1 | 0 | 11 | 01 | 00 |
| 336 | 97 | 18E1FF9 | 02FFFB82 | 189B1EDA2 | 7352C62315 | 1 | 1 | 1 | 0 | 0 | 11 | 11 | 01 |
| 337 | 98 | 11C3FF2 | 05FFF705 | 11363DB45 | 66A58C462B | 0 | 1 | 0 | 1 | 1 | 11 | 11 | 11 |
| 338 | 99 | 0387FE4 | 0BFFEE0A | 026C7B68B | 4D4B188C56 | 0 | 1 | 0 | 0 | 0 | 11 | 11 | 11 |
| 339 | 100 | 070FFC9 | 17FFDC15 | 04D8F6D16 | 1A963118AD | 0 | 1 | 0 | 1 | 1 | 11 | 11 | 11 |
| 340 | 101 | 0E1FF92 | 2FFFB82B | 09B1EDA2C | 352C62315A | 1 | 1 | 1 | 0 | 0 | 10 | 11 | 11 |

*Sample Data*

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 341 | 102 | 1C3FF24 | 5FFF7057 | 1363DB458 | 6A58C462B4 | 1 | 1 | 0 | 0 | 0 | 10 | 10 | 11 |
| 342 | 103 | 187FE48 | 3FFEE0AE | 06C7B68B0 | 54B188C569 | 1 | 1 | 0 | 1 | 1 | 01 | 10 | 10 |
| 343 | 104 | 10FFC90 | 7FFDC15C | 0D8F6D161 | 2963118AD2 | 0 | 1 | 1 | 0 | 1 | 01 | 01 | 10 |
| 344 | 105 | 01FF920 | 7FFB82B9 | 1B1EDA2C2 | 52C62315A5 | 0 | 1 | 1 | 1 | 0 | 00 | 01 | 01 |
| 345 | 106 | 03FF240 | 7FF70573 | 163DB4584 | 258C462B4B | 0 | 1 | 0 | 1 | 0 | 10 | 00 | 01 |
| 346 | 107 | 07FE481 | 7FEE0AE6 | 0C7B68B08 | 4B188C5696 | 0 | 1 | 1 | 0 | 0 | 00 | 10 | 00 |
| 347 | 108 | 0FFC902 | 7FDC15CD | 18F6D1610 | 163118AD2D | 1 | 1 | 1 | 0 | 1 | 00 | 00 | 10 |
| 348 | 109 | 1FF9204 | 7FB82B9A | 11EDA2C20 | 2C62315A5B | 1 | 1 | 0 | 0 | 0 | 01 | 00 | 00 |
| 349 | 110 | 1FF2408 | 7F705735 | 03DB45841 | 58C462B4B6 | 1 | 0 | 0 | 1 | 1 | 00 | 01 | 00 |
| 350 | 111 | 1FE4810 | 7EE0AE6B | 07B68B082 | 3188C5696C | 1 | 1 | 0 | 1 | 1 | 10 | 00 | 01 |
| 351 | 112 | 1FC9021 | 7DC15CD6 | 0F6D16105 | 63118AD2D8 | 1 | 1 | 1 | 0 | 1 | 00 | 10 | 00 |
| 352 | 113 | 1F92042 | 7B82B9AD | 1EDA2C20B | 462315A5B0 | 1 | 1 | 1 | 0 | 1 | 00 | 00 | 10 |
| 353 | 114 | 1F24084 | 7705735A | 1DB458416 | 0C462B4B61 | 1 | 0 | 1 | 0 | 0 | 01 | 00 | 00 |
| 354 | 115 | 1E48108 | 6E0AE6B5 | 1B68B082C | 188C5696C3 | 1 | 0 | 1 | 1 | 0 | 11 | 01 | 00 |
| 355 | 116 | 1C90211 | 5C15CD6A | 16D161059 | 3118AD2D86 | 1 | 0 | 0 | 0 | 0 | 10 | 11 | 01 |
| 356 | 117 | 1920422 | 382B9AD5 | 0DA2C20B3 | 62315A5B0D | 1 | 0 | 1 | 0 | 0 | 10 | 10 | 11 |
| 357 | 118 | 1240845 | 705735AA | 1B4584167 | 4462B4B61A | 0 | 0 | 1 | 0 | 1 | 10 | 10 | 10 |
| 358 | 119 | 048108A | 60AE6B55 | 168B082CF | 08C5696C34 | 0 | 1 | 0 | 1 | 0 | 01 | 10 | 10 |
| 359 | 120 | 0902114 | 415CD6AB | 0D161059E | 118AD2D869 | 1 | 0 | 1 | 1 | 0 | 10 | 01 | 10 |
| 360 | 121 | 1204228 | 02B9AD56 | 1A2C20B3D | 2315A5B0D2 | 0 | 1 | 1 | 0 | 0 | 11 | 10 | 01 |
| 361 | 122 | 0408451 | 05735AAD | 14584167B | 462B4B61A4 | 0 | 0 | 0 | 0 | 1 | 11 | 11 | 10 |
| 362 | 123 | 08108A2 | 0AE6B55B | 08B082CF7 | 0C5696C348 | 1 | 1 | 1 | 0 | 0 | 10 | 11 | 11 |
| 363 | 124 | 1021144 | 15CD6AB6 | 1161059EF | 18AD2D8690 | 0 | 1 | 0 | 1 | 0 | 10 | 10 | 11 |
| 364 | 125 | 0042289 | 2B9AD56C | 02C20B3DE | 315A5B0D20 | 0 | 1 | 0 | 0 | 1 | 10 | 10 | 10 |

## 1.5 FOURTH SET OF SAMPLES

```
Initial values for the key, pan address and clock

K'c4[0]   = 21   K'c4[1]   = 87    K'c4[2]   = F0    K'c4[3]  = 4A
K'c4[4]   = BA   K'c4[5]   = 90    K'c4[6]   = 31    K'c4[7]  = D0
K'c4[8]   = 78   K'c4[9]   = 0D    K'c4[10]  = 4C    K'c4[11] = 53
K'c4[12]  = E0   K'c4[13]  = 15    K'c4[14]  = 3A    K'c4[15] = 63

Addr4[0] = 2C   Addr4[1] = 7F    Addr4[2] = 94
Addr4[3] = 56   Addr4[4] = 0F    Addr4[5] = 1B

CL4[0]  = 5F   CL4[1]  = 1A    CL4[2]  = 00    CL4[3]  = 02


================================================================================================
         Fill LFSRs with initial data
================================================================================================

  t  clk#    LFSR1    LFSR2     LFSR3      LFSR4          X1 X2 X3 X4   Z  C[t+1] C[t] C[t-1]

  0     0    0000000* 00000000* 000000000* 0000000000*    0  0  0  0    0  00     00   00
  1     1    0000000* 00000001* 000000001* 0000000001*    0  0  0  0    0  00     00   00
  2     2    0000001* 00000002* 000000002* 0000000003*    0  0  0  0    0  00     00   00
  3     3    0000002* 00000004* 000000004* 0000000007*    0  0  0  0    0  00     00   00
  4     4    0000004* 00000009* 000000008* 000000000F*    0  0  0  0    0  00     00   00
  5     5    0000008* 00000013* 000000010* 000000001E*    0  0  0  0    0  00     00   00
  6     6    0000010* 00000027* 000000021* 000000003D*    0  0  0  0    0  00     00   00
  7     7    0000021* 0000004F* 000000043* 000000007A*    0  0  0  0    0  00     00   00
  8     8    0000042* 0000009F* 000000087* 00000000F4*    0  0  0  0    0  00     00   00
  9     9    0000084* 0000013F* 00000010F* 00000001E9*    0  0  0  0    0  00     00   00
 10    10    0000108* 0000027F* 00000021F* 00000003D2*    0  0  0  0    0  00     00   00
 11    11    0000211* 000004FE* 00000043E* 00000007A5*    0  0  0  0    0  00     00   00
 12    12    0000422* 000009FC* 00000087C* 0000000F4A*    0  0  0  0    0  00     00   00
 13    13    0000845* 000013F8* 0000010F8* 0000001E94*    0  0  0  0    0  00     00   00
 14    14    000108B* 000027F0* 0000021F1* 0000003D29*    0  0  0  0    0  00     00   00
 15    15    0002117* 00004FE1* 0000043E3* 0000007A52*    0  0  0  0    0  00     00   00
 16    16    000422E* 00009FC2* 0000087C6* 000000F4A4*    0  0  0  0    0  00     00   00
 17    17    000845D* 00013F84* 000010F8C* 000001E948*    0  0  0  0    0  00     00   00
 18    18    00108BA* 00027F08* 000021F18* 000003D290*    0  0  0  0    0  00     00   00
 19    19    0021174* 0004FE10* 000043E30* 000007A520*    0  0  0  0    0  00     00   00
 20    20    00422E8* 0009FC21* 000087C61* 00000F4A41*    0  0  0  0    0  00     00   00
 21    21    00845D1* 0013F842* 00010F8C3* 00001E9482*    0  0  0  0    0  00     00   00
 22    22    0108BA3* 0027F084* 00021F186* 00003D2905*    0  0  0  0    0  00     00   00
 23    23    0211747* 004FE109* 00043E30C* 00007A520B*    0  0  0  0    0  00     00   00
 24    24    0422E8F* 009FC213* 00087C619* 0000F4A417*    0  1  0  0    1  00     00   00
 25    25    0845D1E* 013F8426* 0010F8C32* 0001E9482F*    1  0  0  0    1  00     00   00
 26    26    108BA3D  027F084D* 0021F1864* 0003D2905E*    0  0  0  0    0  00     00   00
 27    27    011747B  04FE109B* 0043E30C9* 0007A520BC*    0  1  0  0    1  00     00   00
 28    28    022E8F6  09FC2136* 0087C6192* 000F4A4179*    0  1  0  0    1  00     00   00
 29    29    045D1EC  13F8426C* 010F8C325* 001E9482F2*    0  1  0  0    1  00     00   00
 30    30    08BA3D9  27F084D8* 021F1864B* 003D2905E5*    1  1  0  0    0  01     00   00
 31    31    11747B3  4FE109B0* 043E30C97* 007A520BCA*    0  1  0  0    1  00     00   00
 32    32    02E8F67  1FC21360  087C6192E* 00F4A41795*    0  1  1  1    1  01     00   00
 33    33    05D1ECF  3F8426C1  10F8C325C* 01E9482F2B*    0  1  0  1    0  01     00   00
 34    34    0BA3D9F  7F084D82  01F1864B8  03D2905E56*    1  0  0  1    0  01     00   00
```

*Sample Data*

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 35 | 35 | 1747B3E | 7E109B04 | 03E30C970 | 07A520BCAC* | 0 | 0 | 0 | 1 | 1 | 00 | 00 | 00 |
| 36 | 36 | 0E8F67C | 7C213608 | 07C6192E1 | 0F4A417958* | 1 | 0 | 0 | 0 | 1 | 00 | 00 | 00 |
| 37 | 37 | 1D1ECF8 | 78426C11 | 0F8C325C3 | 1E9482F2B1* | 1 | 0 | 1 | 1 | 1 | 01 | 00 | 00 |
| 38 | 38 | 1A3D9F0 | 7084D822 | 1F1864B86 | 3D2905E563* | 1 | 1 | 1 | 0 | 1 | 01 | 00 | 00 |
| 39 | 39 | 147B3E1 | 6109B044 | 1E30C970C | 7A520BCAC6* | 0 | 0 | 1 | 0 | 1 | 00 | 00 | 00 |

Start clocking Summation Combiner

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 40 | 1 | 08F67C2 | 42136088 | 1C6192E18 | 74A417958D | 1 | 0 | 1 | 1 | 1 | 01 | 00 | 00 |
| 41 | 2 | 11ECF84 | 0426C111 | 18C325C30 | 69482F2B1B | 0 | 0 | 1 | 0 | 0 | 00 | 01 | 00 |
| 42 | 3 | 03D9F08 | 084D8222 | 11864B861 | 52905E5637 | 0 | 0 | 0 | 1 | 1 | 11 | 00 | 01 |
| 43 | 4 | 07B3E10 | 109B0444 | 030C970C3 | 2520BCAC6E | 0 | 1 | 0 | 0 | 0 | 01 | 11 | 00 |
| 44 | 5 | 0F67C21 | 21360889 | 06192E186 | 4A417958DC | 1 | 0 | 0 | 0 | 0 | 10 | 01 | 11 |
| 45 | 6 | 1ECF843 | 426C1112 | 0C325C30C | 1482F2B1B8 | 1 | 0 | 1 | 1 | 1 | 11 | 10 | 01 |
| 46 | 7 | 1D9F086 | 04D82225 | 1864B8619 | 2905E56370 | 1 | 1 | 1 | 0 | 0 | 01 | 11 | 10 |
| 47 | 8 | 1B3E10D | 09B0444B | 10C970C32 | 520BCAC6E1 | 1 | 1 | 0 | 0 | 1 | 10 | 01 | 11 |
| 48 | 9 | 167C21B | 13608897 | 0192E1865 | 2417958DC3 | 0 | 0 | 0 | 0 | 0 | 00 | 10 | 01 |
| 49 | 10 | 0CF8436 | 26C1112F | 0325C30CB | 482F2B1B87 | 1 | 1 | 0 | 0 | 0 | 00 | 00 | 10 |
| 50 | 11 | 19F086D | 4D82225E | 064B86197 | 105E56370F | 1 | 1 | 0 | 0 | 0 | 01 | 00 | 00 |
| 51 | 12 | 13E10DB | 1B0444BC | 0C970C32F | 20BCAC6E1F | 0 | 0 | 1 | 1 | 1 | 00 | 01 | 00 |
| 52 | 13 | 07C21B7 | 36088979 | 192E1865E | 417958DC3F | 0 | 0 | 1 | 0 | 1 | 11 | 00 | 01 |
| 53 | 14 | 0F8436E | 6C1112F2 | 125C30CBD | 02F2B1B87F | 1 | 0 | 0 | 1 | 1 | 01 | 11 | 00 |
| 54 | 15 | 1F086DD | 582225E4 | 04B86197B | 05E56370FF | 1 | 0 | 0 | 1 | 1 | 10 | 01 | 11 |
| 55 | 16 | 1E10DBA | 30444BC9 | 0970C32F7 | 0BCAC6E1FF | 1 | 0 | 1 | 1 | 1 | 11 | 10 | 01 |
| 56 | 17 | 1C21B75 | 60889793 | 12E1865EE | 17958DC3FF | 1 | 1 | 0 | 1 | 0 | 01 | 11 | 10 |
| 57 | 18 | 18436EA | 41112F27 | 05C30CBDD | 2F2B1B87FF | 1 | 0 | 0 | 0 | 0 | 10 | 01 | 11 |
| 58 | 19 | 1086DD4 | 02225E4E | 0B86197BA | 5E56370FFF | 0 | 0 | 1 | 0 | 1 | 00 | 10 | 01 |
| 59 | 20 | 010DBA8 | 0444BC9D | 170C32F74 | 3CAC6E1FFF | 0 | 0 | 0 | 1 | 1 | 01 | 00 | 10 |
| 60 | 21 | 021B750 | 0889793A | 0E1865EE8 | 7958DC3FFF | 0 | 1 | 1 | 0 | 1 | 00 | 01 | 00 |
| 61 | 22 | 0436EA0 | 1112F274 | 1C30CBDD0 | 72B1B87FFE | 0 | 0 | 1 | 1 | 0 | 10 | 00 | 01 |
| 62 | 23 | 086DD40 | 2225E4E9 | 186197BA1 | 656370FFFC | 1 | 0 | 1 | 0 | 0 | 00 | 10 | 00 |
| 63 | 24 | 10DBA81 | 444BC9D3 | 10C32F743 | 4AC6E1FFF8 | 0 | 0 | 0 | 1 | 1 | 01 | 00 | 10 |
| 64 | 25 | 01B7502 | 089793A7 | 01865EE86 | 158DC3FFF1 | 0 | 1 | 0 | 1 | 1 | 00 | 01 | 00 |
| 65 | 26 | 036EA05 | 112F274E | 030CBDD0D | 2B1B87FFE3 | 0 | 0 | 0 | 0 | 0 | 11 | 00 | 01 |
| 66 | 27 | 06DD40B | 225E4E9C | 06197BA1A | 56370FFFC6 | 0 | 0 | 0 | 0 | 1 | 10 | 11 | 00 |
| 67 | 28 | 0DBA817 | 44BC9D39 | 0C32F7434 | 2C6E1FFF8D | 1 | 1 | 1 | 0 | 1 | 10 | 10 | 11 |
| 68 | 29 | 1B7502E | 09793A72 | 1865EE868 | 58DC3FFF1B | 1 | 0 | 1 | 1 | 1 | 01 | 10 | 10 |
| 69 | 30 | 16EA05D | 12F274E5 | 10CBDD0D0 | 31B87FFE36 | 0 | 1 | 0 | 1 | 1 | 01 | 01 | 10 |
| 70 | 31 | 0DD40BA | 25E4E9CB | 0197BA1A1 | 6370FFFC6D | 1 | 1 | 0 | 0 | 1 | 11 | 01 | 01 |
| 71 | 32 | 1BA8174 | 4BC9D397 | 032F74343 | 46E1FFF8DA | 1 | 1 | 0 | 1 | 0 | 11 | 11 | 01 |
| 72 | 33 | 17502E8 | 1793A72F | 065EE8687 | 0DC3FFF1B4 | 0 | 1 | 0 | 1 | 1 | 11 | 11 | 11 |
| 73 | 34 | 0EA05D0 | 2F274E5E | 0CBDD0D0F | 1B87FFE369 | 1 | 0 | 1 | 1 | 0 | 10 | 11 | 11 |
| 74 | 35 | 1D40BA0 | 5E4E9CBD | 197BA1A1F | 370FFFC6D2 | 1 | 0 | 1 | 0 | 0 | 10 | 10 | 11 |
| 75 | 36 | 1A81741 | 3C9D397B | 12F74343F | 6E1FFF8DA5 | 1 | 1 | 0 | 0 | 0 | 01 | 10 | 10 |
| 76 | 37 | 1502E82 | 793A72F6 | 05EE8687F | 5C3FFF1B4B | 0 | 0 | 0 | 0 | 1 | 00 | 01 | 10 |
| 77 | 38 | 0A05D05 | 7274E5ED | 0BDD0D0FF | 387FFE3696 | 1 | 0 | 1 | 0 | 0 | 10 | 00 | 01 |
| 78 | 39 | 140BA0B | 64E9CBDA | 17BA1A1FF | 70FFFC6D2C | 0 | 1 | 0 | 1 | 0 | 00 | 10 | 00 |
| 79 | 40 | 0817416 | 49D397B4 | 0F74343FE | 61FFF8DA59 | 1 | 1 | 1 | 1 | 0 | 11 | 00 | 10 |
| 80 | 41 | 102E82C | 13A72F69 | 1EE8687FD | 43FFF1B4B3 | 0 | 1 | 1 | 1 | 0 | 00 | 11 | 00 |
| 81 | 42 | 005D058 | 274E5ED2 | 1DD0D0FFA | 07FFE36966 | 0 | 0 | 1 | 1 | 0 | 11 | 00 | 11 |
| 82 | 43 | 00BA0B0 | 4E9CBDA5 | 1BA1A1FF5 | 0FFFC6D2CD | 0 | 1 | 1 | 1 | 0 | 00 | 11 | 00 |
| 83 | 44 | 0174160 | 1D397B4A | 174343FEA | 1FFF8DA59B | 0 | 0 | 0 | 1 | 1 | 10 | 00 | 11 |
| 84 | 45 | 02E82C0 | 3A72F695 | 0E8687FD4 | 3FFF1B4B37 | 0 | 0 | 1 | 1 | 0 | 00 | 10 | 00 |
| 85 | 46 | 05D0580 | 74E5ED2B | 1D0D0FFA9 | 7FFE36966E | 0 | 1 | 1 | 1 | 1 | 00 | 00 | 10 |
| 86 | 47 | 0BA0B00 | 69CBDA56 | 1A1A1FF53 | 7FFC6D2CDC | 1 | 1 | 1 | 1 | 0 | 10 | 00 | 00 |
| 87 | 48 | 1741600 | 5397B4AC | 14343FEA6 | 7FF8DA59B8 | 0 | 1 | 0 | 1 | 0 | 00 | 10 | 00 |
| 88 | 49 | 0E82C01 | 272F6959 | 08687FD4D | 7FF1B4B370 | 1 | 0 | 1 | 1 | 1 | 00 | 00 | 10 |

*Sample Data*

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 89 | 50 | 1D05802 | 4E5ED2B3 | 10D0FFA9A | 7FE36966E0 | 1 | 0 | 0 | 1 | 0 | 01 | 00 | 00 |
| 90 | 51 | 1A0B004 | 1CBDA566 | 01A1FF535 | 7FC6D2CDC0 | 1 | 1 | 0 | 1 | 0 | 11 | 01 | 00 |
| 91 | 52 | 1416009 | 397B4ACC | 0343FEA6B | 7F8DA59B80 | 0 | 0 | 0 | 1 | 0 | 10 | 11 | 01 |
| 92 | 53 | 082C013 | 72F69599 | 0687FD4D7 | 7F1B4B3701 | 1 | 1 | 0 | 0 | 0 | 10 | 10 | 11 |
| 93 | 54 | 1058026 | 65ED2B33 | 0D0FFA9AF | 7E36966E03 | 0 | 1 | 1 | 0 | 0 | 01 | 10 | 10 |
| 94 | 55 | 00B004D | 4BDA5667 | 1A1FF535E | 7C6D2CDC06 | 0 | 1 | 1 | 0 | 1 | 01 | 01 | 10 |
| 95 | 56 | 016009B | 17B4ACCE | 143FEA6BD | 78DA59B80D | 0 | 1 | 0 | 1 | 1 | 11 | 01 | 01 |
| 96 | 57 | 02C0137 | 2F69599D | 087FD4D7B | 71B4B3701A | 0 | 0 | 1 | 1 | 1 | 10 | 11 | 01 |
| 97 | 58 | 058026F | 5ED2B33B | 10FFA9AF6 | 636966E034 | 0 | 1 | 0 | 0 | 1 | 01 | 10 | 11 |
| 98 | 59 | 0B004DF | 3DA56677 | 01FF535ED | 46D2CDC068 | 1 | 1 | 0 | 1 | 0 | 10 | 01 | 10 |
| 99 | 60 | 16009BF | 7B4ACCEF | 03FEA6BDB | 0DA59B80D0 | 0 | 0 | 0 | 1 | 1 | 00 | 10 | 01 |
| 100 | 61 | 0C0137F | 769599DF | 07FD4D7B7 | 1B4B3701A1 | 1 | 1 | 0 | 0 | 0 | 00 | 00 | 10 |
| 101 | 62 | 18026FE | 6D2B33BE | 0FFA9AF6E | 36966E0342 | 1 | 0 | 1 | 1 | 1 | 01 | 00 | 00 |
| 102 | 63 | 1004DFC | 5A56677D | 1FF535EDD | 6D2CDC0684 | 0 | 0 | 1 | 0 | 0 | 00 | 01 | 00 |
| 103 | 64 | 0009BF9 | 34ACCEFB | 1FEA6BDBB | 5A59B80D09 | 0 | 1 | 1 | 0 | 0 | 10 | 00 | 01 |
| 104 | 65 | 00137F2 | 69599DF7 | 1FD4D7B76 | 34B3701A12 | 0 | 0 | 1 | 1 | 0 | 00 | 10 | 00 |
| 105 | 66 | 0026FE5 | 52B33BEF | 1FA9AF6EC | 6966E03424 | 0 | 1 | 1 | 0 | 0 | 00 | 00 | 10 |
| 106 | 67 | 004DFCA | 256677DF | 1F535EDD8 | 52CDC06848 | 0 | 0 | 1 | 1 | 0 | 01 | 00 | 00 |
| 107 | 68 | 009BF94 | 4ACCEFBE | 1EA6BDBB0 | 259B80D091 | 0 | 1 | 1 | 1 | 0 | 11 | 01 | 00 |
| 108 | 69 | 0137F29 | 1599DF7C | 1D4D7B760 | 4B3701A123 | 0 | 1 | 1 | 0 | 1 | 10 | 11 | 01 |
| 109 | 70 | 026FE53 | 2B33BEF9 | 1A9AF6EC0 | 166E034246 | 0 | 0 | 1 | 0 | 1 | 01 | 10 | 11 |
| 110 | 71 | 04DFCA7 | 56677DF2 | 1535EDD81 | 2CDC06848D | 0 | 0 | 0 | 1 | 0 | 01 | 01 | 10 |
| 111 | 72 | 09BF94F | 2CCEFBE4 | 0A6BDBB03 | 59B80D091B | 1 | 1 | 1 | 1 | 1 | 00 | 01 | 01 |
| 112 | 73 | 137F29E | 599DF7C9 | 14D7B7607 | 33701A1236 | 0 | 1 | 0 | 0 | 1 | 11 | 00 | 01 |
| 113 | 74 | 06FE53C | 333BEF93 | 09AF6EC0E | 66E034246C | 0 | 0 | 1 | 1 | 1 | 01 | 11 | 00 |
| 114 | 75 | 0DFCA79 | 6677DF26 | 135EDD81D | 4DC06848D8 | 1 | 0 | 0 | 1 | 1 | 10 | 01 | 11 |
| 115 | 76 | 1BF94F2 | 4CEFBE4D | 06BDBB03B | 1B80D091B1 | 1 | 1 | 0 | 1 | 1 | 11 | 10 | 01 |
| 116 | 77 | 17F29E5 | 19DF7C9A | 0D7B76077 | 3701A12363 | 0 | 1 | 1 | 0 | 1 | 00 | 11 | 10 |
| 117 | 78 | 0FE53CA | 33BEF934 | 1AF6EC0EF | 6E034246C6 | 1 | 1 | 1 | 0 | 1 | 11 | 00 | 11 |
| 118 | 79 | 1FCA794 | 677DF269 | 15EDD81DF | 5C06848D8C | 1 | 0 | 0 | 0 | 0 | 01 | 11 | 00 |
| 119 | 80 | 1F94F29 | 4EFBE4D2 | 0BDBB03BE | 380D091B19 | 1 | 1 | 1 | 0 | 0 | 01 | 01 | 11 |
| 120 | 81 | 1F29E53 | 1DF7C9A5 | 17B76077D | 701A123633 | 1 | 1 | 0 | 0 | 1 | 11 | 01 | 01 |
| 121 | 82 | 1E53CA6 | 3BEF934B | 0F6EC0EFB | 6034246C66 | 1 | 1 | 1 | 0 | 0 | 11 | 11 | 01 |
| 122 | 83 | 1CA794D | 77DF2696 | 1EDD81DF6 | 406848D8CD | 1 | 1 | 1 | 0 | 0 | 10 | 11 | 11 |
| 123 | 84 | 194F29B | 6FBE4D2C | 1DBB03BED | 00D091B19B | 1 | 1 | 1 | 1 | 0 | 11 | 10 | 11 |
| 124 | 85 | 129E536 | 5F7C9A59 | 1B76077DA | 01A1236337 | 0 | 0 | 1 | 1 | 1 | 00 | 11 | 10 |
| 125 | 86 | 053CA6C | 3EF934B3 | 16EC0EFB4 | 034246C66E | 0 | 1 | 0 | 0 | 1 | 10 | 00 | 11 |
| 126 | 87 | 0A794D9 | 7DF26967 | 0DD81DF69 | 06848D8CDD | 1 | 1 | 1 | 1 | 0 | 01 | 10 | 00 |
| 127 | 88 | 14F29B3 | 7BE4D2CF | 1BB03BED3 | 0D091B19BB | 0 | 1 | 1 | 0 | 1 | 01 | 01 | 10 |
| 128 | 89 | 09E5366 | 77C9A59F | 176077DA6 | 1A12363377 | 1 | 1 | 0 | 0 | 1 | 11 | 01 | 01 |
| 129 | 90 | 13CA6CD | 6F934B3F | 0EC0EFB4D | 34246C66EF | 0 | 1 | 1 | 0 | 1 | 10 | 11 | 01 |
| 130 | 91 | 0794D9B | 5F26967F | 1D81DF69A | 6848D8CDDF | 0 | 0 | 1 | 0 | 1 | 01 | 10 | 11 |
| 131 | 92 | 0F29B37 | 3E4D2CFE | 1B03BED35 | 5091B19BBE | 1 | 0 | 1 | 1 | 0 | 10 | 01 | 10 |
| 132 | 93 | 1E5366F | 7C9A59FD | 16077DA6B | 212363377C | 1 | 1 | 0 | 0 | 0 | 11 | 10 | 01 |
| 133 | 94 | 1CA6CDF | 7934B3FB | 0C0EFB4D6 | 4246C66EF9 | 1 | 0 | 1 | 0 | 1 | 00 | 11 | 10 |
| 134 | 95 | 194D9BE | 726967F6 | 181DF69AD | 048D8CDDF2 | 1 | 0 | 1 | 1 | 1 | 11 | 00 | 11 |
| 135 | 96 | 129B37D | 64D2CFED | 103BED35B | 091B19BBE5 | 0 | 1 | 0 | 0 | 0 | 01 | 11 | 00 |
| 136 | 97 | 05366FA | 49A59FDA | 0077DA6B7 | 12363377CA | 0 | 1 | 0 | 0 | 0 | 10 | 01 | 11 |
| 137 | 98 | 0A6CDF5 | 134B3FB4 | 00EFB4D6E | 246C66EF95 | 1 | 0 | 0 | 0 | 1 | 00 | 10 | 01 |
| 138 | 99 | 14D9BEA | 26967F69 | 01DF69ADD | 48D8CDDF2B | 0 | 1 | 0 | 1 | 0 | 00 | 00 | 10 |
| 139 | 100 | 09B37D4 | 4D2CFED2 | 03BED35BB | 11B19BBE56 | 1 | 0 | 0 | 1 | 0 | 01 | 00 | 00 |
| 140 | 101 | 1366FA8 | 1A59FDA5 | 077DA6B77 | 2363377CAC | 0 | 0 | 0 | 0 | 1 | 01 | 01 | 00 |
| 141 | 102 | 06CDF51 | 34B3FB4A | 0EFB4D6EF | 46C66EF959 | 0 | 1 | 1 | 1 | 0 | 00 | 01 | 01 |
| 142 | 103 | 0D9BEA2 | 6967F695 | 1DF69ADDF | 0D8CDDF2B2 | 1 | 0 | 1 | 1 | 1 | 10 | 00 | 01 |
| 143 | 104 | 1B37D45 | 52CFED2A | 1BED35BBF | 1B19BBE564 | 1 | 1 | 1 | 0 | 1 | 00 | 10 | 00 |
| 144 | 105 | 166FA8A | 259FDA54 | 17DA6B77E | 363377CAC8 | 0 | 1 | 0 | 0 | 1 | 01 | 00 | 10 |
| 145 | 106 | 0CDF515 | 4B3FB4A9 | 0FB4D6EFC | 6C66EF9591 | 1 | 0 | 1 | 0 | 1 | 00 | 01 | 00 |

*Sample Data*

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 146 | 107 | 19BEA2B | 167F6952 | 1F69ADDF8 | 58CDDF2B22 | 1 | 0 | 1 | 1 | 1 | 10 | 00 | 01 |
| 147 | 108 | 137D457 | 2CFED2A5 | 1ED35BBF1 | 319BBE5645 | 0 | 1 | 1 | 1 | 1 | 00 | 10 | 00 |
| 148 | 109 | 06FA8AF | 59FDA54A | 1DA6B77E2 | 63377CAC8B | 0 | 1 | 1 | 0 | 0 | 00 | 00 | 10 |
| 149 | 110 | 0DF515F | 33FB4A95 | 1B4D6EFC4 | 466EF95916 | 1 | 1 | 1 | 0 | 1 | 01 | 00 | 00 |
| 150 | 111 | 1BEA2BF | 67F6952A | 169ADDF88 | 0CDDF2B22C | 1 | 1 | 0 | 1 | 0 | 11 | 01 | 00 |
| 151 | 112 | 17D457F | 4FED2A55 | 0D35BBF10 | 19BBE56459 | 0 | 1 | 1 | 1 | 0 | 11 | 11 | 01 |
| 152 | 113 | 0FA8AFE | 1FDA54AB | 1A6B77E20 | 3377CAC8B3 | 1 | 1 | 1 | 0 | 0 | 10 | 11 | 11 |
| 153 | 114 | 1F515FD | 3FB4A957 | 14D6EFC40 | 66EF959166 | 1 | 1 | 0 | 1 | 1 | 10 | 10 | 11 |
| 154 | 115 | 1EA2BFA | 7F6952AF | 09ADDF880 | 4DDF2B22CC | 1 | 0 | 1 | 1 | 1 | 01 | 10 | 10 |
| 155 | 116 | 1D457F4 | 7ED2A55F | 135BBF100 | 1BBE564598 | 1 | 1 | 0 | 1 | 0 | 10 | 01 | 10 |
| 156 | 117 | 1A8AFE8 | 7DA54ABF | 06B77E200 | 377CAC8B31 | 1 | 1 | 0 | 0 | 0 | 11 | 10 | 01 |
| 157 | 118 | 1515FD0 | 7B4A957F | 0D6EFC401 | 6EF9591663 | 0 | 0 | 1 | 1 | 1 | 00 | 11 | 10 |
| 158 | 119 | 0A2BFA1 | 76952AFE | 1ADDF8803 | 5DF2B22CC7 | 1 | 1 | 1 | 1 | 0 | 00 | 00 | 11 |
| 159 | 120 | 1457F42 | 6D2A55FD | 15BBF1007 | 3BE564598E | 0 | 0 | 0 | 1 | 1 | 00 | 00 | 00 |
| 160 | 121 | 08AFE84 | 5A54ABFB | 0B77E200F | 77CAC8B31C | 1 | 0 | 1 | 1 | 1 | 01 | 00 | 00 |
| 161 | 122 | 115FD09 | 34A957F7 | 16EFC401F | 6F95916639 | 0 | 1 | 0 | 1 | 1 | 00 | 01 | 00 |
| 162 | 123 | 02BFA12 | 6952AFEF | 0DDF8803E | 5F2B22CC73 | 0 | 0 | 1 | 0 | 1 | 11 | 00 | 01 |
| 163 | 124 | 057F424 | 52A55FDF | 1BBF1007D | 3E564598E7 | 0 | 1 | 1 | 0 | 1 | 01 | 11 | 00 |
| 164 | 125 | 0AFE848 | 254ABFBF | 177E200FA | 7CAC8B31CF | 1 | 0 | 0 | 1 | 1 | 10 | 01 | 11 |
| 165 | 126 | 15FD090 | 4A957F7E | 0EFC401F5 | 795916639E | 0 | 1 | 1 | 0 | 0 | 11 | 10 | 01 |
| 166 | 127 | 0BFA121 | 152AFEFD | 1DF8803EA | 72B22CC73C | 1 | 0 | 1 | 1 | 0 | 01 | 11 | 10 |
| 167 | 128 | 17F4243 | 2A55FDFA | 1BF1007D4 | 6564598E78 | 0 | 0 | 1 | 0 | 0 | 10 | 01 | 11 |
| 168 | 129 | 0FE8486 | 54ABFBF4 | 17E200FA8 | 4AC8B31CF0 | 1 | 1 | 0 | 1 | 1 | 11 | 10 | 01 |
| 169 | 130 | 1FD090C | 2957F7E8 | 0FC401F51 | 15916639E1 | 1 | 0 | 1 | 1 | 0 | 01 | 11 | 10 |
| 170 | 131 | 1FA1219 | 52AFEFD1 | 1F8803EA3 | 2B22CC73C2 | 1 | 1 | 1 | 0 | 0 | 01 | 01 | 11 |
| 171 | 132 | 1F42432 | 255FDFA2 | 1F1007D47 | 564598E785 | 1 | 0 | 1 | 0 | 1 | 11 | 01 | 01 |
| 172 | 133 | 1E84865 | 4ABFBF44 | 1E200FA8F | 2C8B31CF0B | 1 | 1 | 1 | 1 | 1 | 11 | 11 | 01 |
| 173 | 134 | 1D090CB | 157F7E88 | 1C401F51E | 5916639E17 | 1 | 0 | 1 | 0 | 1 | 11 | 11 | 11 |
| 174 | 135 | 1A12196 | 2AFEFD11 | 18803EA3C | 322CC73C2E | 1 | 1 | 1 | 0 | 0 | 10 | 11 | 11 |
| 175 | 136 | 142432C | 55FDFA23 | 11007D479 | 64598E785C | 0 | 1 | 0 | 0 | 1 | 01 | 10 | 11 |
| 176 | 137 | 0848659 | 2BFBF446 | 0200FA8F2 | 48B31CF0B9 | 1 | 1 | 0 | 1 | 0 | 10 | 01 | 10 |
| 177 | 138 | 1090CB2 | 57F7E88C | 0401F51E4 | 116639E173 | 0 | 1 | 0 | 0 | 1 | 00 | 10 | 01 |
| 178 | 139 | 0121964 | 2FEFD118 | 0803EA3C8 | 22CC73C2E6 | 0 | 1 | 1 | 1 | 1 | 00 | 00 | 10 |
| 179 | 140 | 02432C9 | 5FDFA230 | 1007D4791 | 4598E785CD | 0 | 1 | 0 | 1 | 0 | 01 | 00 | 00 |
| 180 | 141 | 0486593 | 3FBF4461 | 000FA8F23 | 0B31CF0B9B | 0 | 1 | 0 | 0 | 0 | 00 | 01 | 00 |
| 181 | 142 | 090CB26 | 7F7E88C3 | 001F51E47 | 16639E1736 | 1 | 0 | 0 | 0 | 1 | 11 | 00 | 01 |
| 182 | 143 | 121964D | 7EFD1187 | 003EA3C8F | 2CC73C2E6C | 0 | 1 | 0 | 1 | 1 | 01 | 11 | 00 |
| 183 | 144 | 0432C9B | 7DFA230E | 007D4791E | 598E785CD8 | 0 | 1 | 0 | 1 | 1 | 10 | 01 | 11 |
| 184 | 145 | 0865936 | 7BF4461C | 00FA8F23C | 331CF0B9B0 | 1 | 1 | 0 | 0 | 0 | 11 | 10 | 01 |
| 185 | 146 | 10CB26D | 77E88C38 | 01F51E479 | 6639E17361 | 0 | 1 | 0 | 0 | 0 | 00 | 11 | 10 |
| 186 | 147 | 01964DA | 6FD11870 | 03EA3C8F2 | 4C73C2E6C2 | 0 | 1 | 0 | 0 | 1 | 10 | 00 | 11 |
| 187 | 148 | 032C9B4 | 5FA230E1 | 07D4791E4 | 18E785CD84 | 0 | 1 | 0 | 1 | 0 | 00 | 10 | 00 |
| 188 | 149 | 0659368 | 3F4461C2 | 0FA8F23C9 | 31CF0B9B09 | 0 | 0 | 1 | 1 | 0 | 00 | 00 | 10 |
| 189 | 150 | 0CB26D0 | 7E88C384 | 1F51E4793 | 639E173612 | 1 | 1 | 1 | 1 | 0 | 10 | 00 | 00 |
| 190 | 151 | 1964DA0 | 7D118709 | 1EA3C8F27 | 473C2E6C24 | 1 | 0 | 1 | 0 | 0 | 00 | 10 | 00 |
| 191 | 152 | 12C9B41 | 7A230E12 | 1D4791E4E | 0E785CD848 | 0 | 0 | 1 | 0 | 1 | 01 | 00 | 10 |
| 192 | 153 | 0593683 | 74461C24 | 1A8F23C9C | 1CF0B9B091 | 0 | 0 | 1 | 1 | 1 | 00 | 01 | 00 |
| 193 | 154 | 0B26D06 | 688C3848 | 151E47938 | 39E1736123 | 1 | 1 | 0 | 1 | 1 | 10 | 00 | 01 |
| 194 | 155 | 164DA0D | 51187091 | 0A3C8F271 | 73C2E6C247 | 0 | 0 | 1 | 1 | 0 | 00 | 10 | 00 |
| 195 | 156 | 0C9B41A | 2230E123 | 14791E4E3 | 6785CD848F | 1 | 0 | 0 | 1 | 0 | 00 | 00 | 10 |
| 196 | 157 | 1936835 | 4461C247 | 08F23C9C6 | 4F0B9B091E | 1 | 0 | 1 | 0 | 0 | 01 | 00 | 00 |
| 197 | 158 | 126D06A | 08C3848E | 11E47938D | 1E1736123C | 0 | 1 | 0 | 0 | 0 | 00 | 01 | 00 |
| 198 | 159 | 04DA0D5 | 1187091C | 03C8F271B | 3C2E6C2478 | 0 | 1 | 0 | 0 | 1 | 11 | 00 | 01 |
| 199 | 160 | 09B41AA | 230E1238 | 0791E4E37 | 785CD848F1 | 1 | 0 | 0 | 0 | 0 | 01 | 11 | 00 |
| 200 | 161 | 1368354 | 461C2470 | 0F23C9C6F | 70B9B091E3 | 0 | 0 | 1 | 1 | 1 | 10 | 01 | 11 |
| 201 | 162 | 06D06A9 | 0C3848E1 | 1E47938DF | 61736123C6 | 0 | 0 | 1 | 0 | 1 | 00 | 10 | 01 |
| 202 | 163 | 0DA0D52 | 187091C3 | 1C8F271BE | 42E6C2478D | 1 | 0 | 1 | 1 | 1 | 00 | 00 | 10 |

*Sample Data*

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 203 | 164 | 1B41AA4 | 30E12387 | 191E4E37C | 05CD848F1A | 1 | 1 | 1 | 1 | 0 | 10 | 00 | 00 |
| 204 | 165 | 1683549 | 61C2470F | 123C9C6F9 | 0B9B091E34 | 0 | 1 | 0 | 1 | 0 | 00 | 10 | 00 |
| 205 | 166 | 0D06A92 | 43848E1E | 047938DF3 | 1736123C68 | 1 | 1 | 0 | 0 | 0 | 00 | 00 | 10 |
| 206 | 167 | 1A0D524 | 07091C3C | 08F271BE7 | 2E6C2478D1 | 1 | 0 | 1 | 0 | 0 | 01 | 00 | 00 |
| 207 | 168 | 141AA49 | 0E123879 | 11E4E37CF | 5CD848F1A2 | 0 | 0 | 0 | 1 | 0 | 00 | 01 | 00 |
| 208 | 169 | 0835492 | 1C2470F3 | 03C9C6F9F | 39B091E345 | 1 | 0 | 0 | 1 | 0 | 10 | 00 | 01 |
| 209 | 170 | 106A925 | 3848E1E6 | 07938DF3F | 736123C68B | 0 | 0 | 0 | 0 | 0 | 11 | 10 | 00 |
| 210 | 171 | 00D524A | 7091C3CD | 0F271BE7E | 66C2478D16 | 0 | 1 | 1 | 1 | 0 | 01 | 11 | 10 |
| 211 | 172 | 01AA495 | 6123879B | 1E4E37CFD | 4D848F1A2D | 0 | 0 | 1 | 1 | 1 | 10 | 01 | 11 |
| 212 | 173 | 035492A | 42470F36 | 1C9C6F9FB | 1B091E345B | 0 | 0 | 1 | 0 | 1 | 00 | 10 | 01 |
| 213 | 174 | 06A9255 | 048E1E6C | 1938DF3F6 | 36123C68B7 | 0 | 1 | 1 | 0 | 0 | 00 | 00 | 10 |
| 214 | 175 | 0D524AB | 091C3CD8 | 1271BE7EC | 6C2478D16E | 1 | 0 | 0 | 0 | 1 | 00 | 00 | 00 |
| 215 | 176 | 1AA4957 | 123879B1 | 04E37CFD8 | 5848F1A2DD | 1 | 0 | 0 | 0 | 1 | 00 | 00 | 00 |
| 216 | 177 | 15492AF | 2470F363 | 09C6F9FB0 | 3091E345BA | 0 | 0 | 1 | 1 | 0 | 01 | 00 | 00 |
| 217 | 178 | 0A9255E | 48E1E6C7 | 138DF3F61 | 6123C68B75 | 1 | 1 | 0 | 0 | 1 | 00 | 01 | 00 |
| 218 | 179 | 1524ABD | 11C3CD8F | 071BE7EC3 | 42478D16EB | 0 | 1 | 0 | 0 | 1 | 11 | 00 | 01 |
| 219 | 180 | 0A4957B | 23879B1F | 0E37CFD87 | 048F1A2DD6 | 1 | 1 | 1 | 1 | 1 | 00 | 11 | 00 |
| 220 | 181 | 1492AF6 | 470F363F | 1C6F9FB0E | 091E345BAD | 0 | 0 | 1 | 0 | 1 | 10 | 00 | 11 |
| 221 | 182 | 09255EC | 0E1E6C7F | 18DF3F61D | 123C68B75B | 1 | 0 | 1 | 0 | 0 | 00 | 10 | 00 |
| 222 | 183 | 124ABD9 | 1C3CD8FF | 11BE7EC3A | 2478D16EB6 | 0 | 0 | 0 | 0 | 0 | 01 | 00 | 10 |
| 223 | 184 | 04957B3 | 3879B1FE | 037CFD874 | 48F1A2DD6D | 0 | 0 | 0 | 1 | 0 | 00 | 01 | 00 |
| 224 | 185 | 092AF66 | 70F363FD | 06F9FB0E9 | 11E345BADB | 1 | 1 | 0 | 1 | 1 | 10 | 00 | 01 |
| 225 | 186 | 1255ECD | 61E6C7FA | 0DF3F61D3 | 23C68B75B7 | 0 | 1 | 1 | 1 | 1 | 00 | 10 | 00 |
| 226 | 187 | 04ABD9B | 43CD8FF5 | 1BE7EC3A7 | 478D16EB6E | 0 | 1 | 1 | 1 | 1 | 00 | 00 | 10 |
| 227 | 188 | 0957B37 | 079B1FEA | 17CFD874E | 0F1A2DD6DD | 1 | 1 | 0 | 0 | 0 | 01 | 00 | 00 |
| 228 | 189 | 12AF66F | 0F363FD4 | 0F9FB0E9C | 1E345BADBB | 0 | 0 | 1 | 0 | 0 | 00 | 01 | 00 |
| 229 | 190 | 055ECDE | 1E6C7FA9 | 1F3F61D39 | 3C68B75B76 | 0 | 0 | 1 | 0 | 1 | 11 | 00 | 01 |
| 230 | 191 | 0ABD9BC | 3CD8FF53 | 1E7EC3A73 | 78D16EB6EC | 1 | 1 | 1 | 1 | 1 | 00 | 11 | 00 |
| 231 | 192 | 157B379 | 79B1FEA7 | 1CFD874E6 | 71A2DD6DD9 | 0 | 1 | 1 | 1 | 1 | 11 | 00 | 11 |
| 232 | 193 | 0AF66F3 | 7363FD4E | 19FB0E9CD | 6345BADBB2 | 1 | 0 | 1 | 0 | 1 | 01 | 11 | 00 |
| 233 | 194 | 15ECDE6 | 66C7FA9D | 13F61D39A | 468B75B765 | 0 | 1 | 0 | 1 | 1 | 10 | 01 | 11 |
| 234 | 195 | 0BD9BCC | 4D8FF53A | 07EC3A735 | 0D16EB6ECA | 1 | 1 | 0 | 0 | 0 | 11 | 10 | 01 |
| 235 | 196 | 17B3799 | 1B1FEA75 | 0FD874E6A | 1A2DD6DD94 | 0 | 0 | 1 | 0 | 0 | 00 | 11 | 10 |
| 236 | 197 | 0F66F33 | 363FD4EA | 1FB0E9CD5 | 345BADBB28 | 1 | 0 | 1 | 0 | 0 | 11 | 00 | 11 |
| 237 | 198 | 1ECDE67 | 6C7FA9D5 | 1F61D39AA | 68B75B7650 | 1 | 0 | 1 | 1 | 0 | 00 | 11 | 00 |
| 238 | 199 | 1D9BCCF | 58FF53AB | 1EC3A7354 | 516EB6ECA0 | 1 | 1 | 1 | 0 | 1 | 11 | 00 | 11 |
| 239 | 200 | 1B3799E | 31FEA756 | 1D874E6A8 | 22DD6DD940 | 1 | 1 | 1 | 1 | 1 | 00 | 11 | 00 |

```
Z[0]  = 3F
Z[1]  = B1
Z[2]  = 67
Z[3]  = D2
Z[4]  = 2F
Z[5]  = A6
Z[6]  = 1F
Z[7]  = B9
Z[8]  = E6
Z[9]  = 84
Z[10] = 43
Z[11] = 07
Z[12] = D8
Z[13] = 1E
Z[14] = E7
Z[15] = C3
```

*Sample Data*

```
================================================================================================
         Reload this pattern into the LFSRs
         Hold content of Summation Combiner regs and calculate new C[t+1] and Z values
================================================================================================
LFSR1  <= 0E62F3F
LFSR2  <= 6C84A6B1
LFSR3  <= 11E431F67
LFSR4  <= 61E707B9D2
C[t+1] <= 00

================================================================================================
         Generating 125 key symbols (encryption/decryption sequence)
================================================================================================
240     1    0E62F3F  6C84A6B1  11E431F67  61E707B9D2      1  1  0  1    0  00     11   00
241     2    1CC5E7F  59094D63  03C863ECE  43CE0F73A5      1  0  0  1    0  11     00   11
242     3    198BCFF  32129AC6  0790C7D9D  079C1EE74A      1  0  0  1    1  01     11   00
243     4    13179FE  6425358C  0F218FB3A  0F383DCE94      0  0  1  0    0  10     01   11
244     5    062F3FD  484A6B19  1E431F675  1E707B9D28      0  0  1  0    1  00     10   01
245     6    0C5E7FB  1094D632  1C863ECEB  3CE0F73A50      1  1  1  1    0  11     00   10
246     7    18BCFF7  2129AC64  190C7D9D7  79C1EE74A1      1  0  1  1    0  00     11   00
247     8    1179FEE  425358C8  1218FB3AE  7383DCE942      0  0  0  1    1  10     00   11
248     9    02F3FDD  04A6B190  0431F675D  6707B9D285      0  1  0  0    1  11     10   00
249    10    05E7FBB  094D6320  0863ECEBB  4E0F73A50B      0  0  1  0    0  00     11   10
250    11    0BCFF77  129AC640  10C7D9D77  1C1EE74A16      1  1  0  0    0  11     00   11
251    12    179FEEE  25358C80  018FB3AEE  383DCE942C      0  0  0  0    1  10     11   00
252    13    0F3FDDC  4A6B1900  031F675DD  707B9D2859      1  0  0  0    1  01     10   11
253    14    1E7FBB8  14D63200  063ECEBBA  60F73A50B3      1  1  0  1    0  10     01   10
254    15    1CFF771  29AC6401  0C7D9D774  41EE74A167      1  1  1  1    0  10     10   01
255    16    19FEEE2  5358C803  18FB3AEE9  03DCE942CE      1  0  1  1    1  01     10   10
256    17    13FDDC4  26B19007  11F675DD2  07B9D2859C      0  1  0  1    1  01     01   10
257    18    07FBB88  4D63200E  03ECEBBA4  0F73A50B38      0  0  0  0    1  10     01   01
258    19    0FF7711  1AC6401D  07D9D7748  1EE74A1670      1  1  0  1    1  11     10   01
259    20    1FEEE23  358C803B  0FB3AEE91  3DCE942CE1      1  1  1  1    1  01     11   10
260    21    1FDDC47  6B190076  1F675DD23  7B9D2859C2      1  0  1  1    0  01     01   11
261    22    1FBB88F  563200ED  1ECEBBA47  773A50B385      1  0  1  0    1  11     01   01
262    23    1F7711E  2C6401DB  1D9D7748F  6E74A1670A      1  0  1  0    1  10     11   01
263    24    1EEE23D  58C803B6  1B3AEE91E  5CE942CE15      1  1  1  1    0  11     10   11
264    25    1DDC47A  3190076C  1675DD23D  39D2859C2B      1  1  0  1    0  01     11   10
265    26    1BB88F4  63200ED9  0CEBBA47A  73A50B3856      1  0  1  1    0  01     01   11
266    27    17711E8  46401DB2  19D7748F5  674A1670AD      0  0  1  0    0  11     01   01
267    28    0EE23D0  0C803B64  13AEE91EA  4E942CE15B      1  1  0  1    0  11     11   01
268    29    1DC47A0  190076C8  075DD23D4  1D2859C2B7      1  0  0  0    0  11     11   11
269    30    1B88F41  3200ED90  0EBBA47A9  3A50B3856E      1  0  1  0    1  11     11   11
270    31    1711E83  6401DB20  1D7748F53  74A1670ADC      0  0  1  1    1  11     11   11
271    32    0E23D07  4803B641  1AEE91EA7  6942CE15B8      1  0  1  0    1  11     11   11
272    33    1C47A0F  10076C82  15DD23D4F  52859C2B71      1  0  0  1    1  11     11   11
273    34    188F41E  200ED905  0BBA47A9E  250B3856E3      1  0  1  0    1  11     11   11
274    35    111E83C  401DB20A  17748F53D  4A1670ADC7      0  0  0  0    1  00     11   11
275    36    023D078  003B6414  0EE91EA7A  142CE15B8E      0  0  1  0    1  10     00   11
276    37    047A0F0  0076C828  1DD23D4F5  2859C2B71C      0  0  1  0    1  11     10   00
277    38    08F41E1  00ED9050  1BA47A9EA  50B3856E39      1  1  1  1    1  01     11   10
278    39    11E83C2  01DB20A0  1748F53D5  21670ADC72      0  1  0  0    0  10     01   11
279    40    03D0785  03B64141  0E91EA7AA  42CE15B8E4      0  1  1  1    1  11     10   01
280    41    07A0F0A  076C8283  1D23D4F54  059C2B71C8      0  0  1  1    1  00     11   10
281    42    0F41E14  0ED90507  1A47A9EA9  0B3856E390      1  1  1  0    1  11     00   11
282    43    1E83C29  1DB20A0F  148F53D52  1670ADC720      1  1  0  0    1  01     11   00
283    44    1D07853  3B64141E  091EA7AA5  2CE15B8E40      1  0  1  1    0  01     01   11
```

*Sample Data*

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 284 | 45 | 1A0F0A6 | 76C8283C | 123D4F54B | 59C2B71C81 | 1 | 1 | 0 | 1 | 0 | 00 | 01 | 01 |
| 285 | 46 | 141E14C | 6D905079 | 047A9EA97 | 33856E3902 | 0 | 1 | 0 | 1 | 0 | 10 | 00 | 01 |
| 286 | 47 | 083C299 | 5B20A0F2 | 08F53D52F | 670ADC7204 | 1 | 0 | 1 | 0 | 0 | 00 | 10 | 00 |
| 287 | 48 | 1078533 | 364141E4 | 11EA7AA5E | 4E15B8E408 | 0 | 0 | 0 | 0 | 0 | 01 | 00 | 10 |
| 288 | 49 | 00F0A67 | 6C8283C8 | 03D4F54BC | 1C2B71C811 | 0 | 1 | 0 | 0 | 0 | 00 | 01 | 00 |
| 289 | 50 | 01E14CE | 59050791 | 07A9EA978 | 3856E39022 | 0 | 0 | 0 | 0 | 0 | 11 | 00 | 01 |
| 290 | 51 | 03C299C | 320A0F23 | 0F53D52F1 | 70ADC72045 | 0 | 0 | 1 | 1 | 1 | 01 | 11 | 00 |
| 291 | 52 | 0785339 | 64141E47 | 1EA7AA5E2 | 615B8E408A | 0 | 0 | 1 | 0 | 0 | 10 | 01 | 11 |
| 292 | 53 | 0F0A673 | 48283C8E | 1D4F54BC4 | 42B71C8115 | 1 | 0 | 1 | 1 | 1 | 11 | 10 | 01 |
| 293 | 54 | 1E14CE6 | 1050791C | 1A9EA9788 | 056E39022B | 1 | 0 | 1 | 0 | 1 | 00 | 11 | 10 |
| 294 | 55 | 1C299CD | 20A0F239 | 153D52F10 | 0ADC720456 | 1 | 1 | 0 | 1 | 1 | 11 | 00 | 11 |
| 295 | 56 | 185339B | 4141E472 | 0A7AA5E20 | 15B8E408AC | 1 | 0 | 1 | 1 | 0 | 00 | 11 | 00 |
| 296 | 57 | 10A6736 | 0283C8E4 | 14F54BC41 | 2B71C81158 | 0 | 1 | 0 | 0 | 1 | 10 | 00 | 11 |
| 297 | 58 | 014CE6C | 050791C9 | 09EA97882 | 56E39022B0 | 0 | 0 | 1 | 1 | 0 | 00 | 10 | 00 |
| 298 | 59 | 0299CD9 | 0A0F2393 | 13D52F104 | 2DC7204561 | 0 | 0 | 0 | 1 | 1 | 01 | 00 | 10 |
| 299 | 60 | 05339B3 | 141E4726 | 07AA5E208 | 5B8E408AC3 | 0 | 0 | 0 | 1 | 0 | 00 | 01 | 00 |
| 300 | 61 | 0A67366 | 283C8E4C | 0F54BC411 | 371C811587 | 1 | 0 | 1 | 0 | 0 | 10 | 00 | 01 |
| 301 | 62 | 14CE6CC | 50791C98 | 1EA978822 | 6E39022B0F | 0 | 0 | 1 | 0 | 1 | 11 | 10 | 00 |
| 302 | 63 | 099CD99 | 20F23930 | 1D52F1045 | 5C7204561E | 1 | 1 | 1 | 0 | 0 | 01 | 11 | 10 |
| 303 | 64 | 1339B33 | 41E47260 | 1AA5E208B | 38E408AC3D | 0 | 1 | 1 | 1 | 0 | 01 | 01 | 11 |
| 304 | 65 | 0673666 | 03C8E4C0 | 154BC4117 | 71C811587A | 0 | 1 | 0 | 1 | 1 | 11 | 01 | 01 |
| 305 | 66 | 0CE6CCC | 0791C980 | 0A978822E | 639022B0F5 | 1 | 1 | 1 | 1 | 1 | 11 | 11 | 01 |
| 306 | 67 | 19CD999 | 0F239301 | 152F1045C | 47204561EB | 1 | 0 | 0 | 0 | 0 | 11 | 11 | 11 |
| 307 | 68 | 139B332 | 1E472603 | 0A5E208B9 | 0E408AC3D6 | 0 | 0 | 1 | 0 | 0 | 11 | 11 | 11 |
| 308 | 69 | 0736664 | 3C8E4C06 | 14BC41172 | 1C811587AD | 0 | 1 | 0 | 1 | 1 | 11 | 11 | 11 |
| 309 | 70 | 0E6CCC8 | 791C980C | 0978822E5 | 39022B0F5A | 1 | 0 | 1 | 0 | 1 | 11 | 11 | 11 |
| 310 | 71 | 1CD9990 | 72393019 | 12F1045CB | 7204561EB4 | 1 | 0 | 0 | 0 | 0 | 11 | 11 | 11 |
| 311 | 72 | 19B3320 | 64726033 | 05E208B97 | 6408AC3D69 | 1 | 0 | 0 | 0 | 0 | 11 | 11 | 11 |
| 312 | 73 | 1366640 | 48E4C067 | 0BC41172F | 4811587AD3 | 0 | 1 | 1 | 0 | 1 | 11 | 11 | 11 |
| 313 | 74 | 06CCC81 | 11C980CF | 178822E5E | 1022B0F5A6 | 0 | 1 | 0 | 0 | 0 | 11 | 11 | 11 |
| 314 | 75 | 0D99903 | 2393019E | 0F1045CBC | 204561EB4C | 1 | 1 | 1 | 0 | 0 | 10 | 11 | 11 |
| 315 | 76 | 1B33206 | 4726033D | 1E208B979 | 408AC3D699 | 1 | 0 | 1 | 1 | 1 | 10 | 10 | 11 |
| 316 | 77 | 166640D | 0E4C067B | 1C41172F2 | 011587AD33 | 0 | 0 | 1 | 0 | 1 | 10 | 10 | 10 |
| 317 | 78 | 0CCC81B | 1C980CF6 | 18822E5E5 | 022B0F5A66 | 1 | 1 | 1 | 0 | 1 | 01 | 10 | 10 |
| 318 | 79 | 1999036 | 393019EC | 11045CBCA | 04561EB4CD | 1 | 0 | 0 | 0 | 0 | 01 | 01 | 10 |
| 319 | 80 | 133206C | 726033D9 | 0208B9794 | 08AC3D699B | 0 | 0 | 0 | 1 | 0 | 11 | 01 | 01 |
| 320 | 81 | 06640D9 | 64C067B3 | 041172F29 | 11587AD337 | 0 | 1 | 0 | 0 | 0 | 10 | 11 | 01 |
| 321 | 82 | 0CC81B3 | 4980CF66 | 0822E5E53 | 22B0F5A66F | 1 | 1 | 1 | 1 | 0 | 11 | 10 | 11 |
| 322 | 83 | 1990366 | 13019ECC | 1045CBCA6 | 4561EB4CDF | 1 | 0 | 0 | 0 | 0 | 00 | 11 | 10 |
| 323 | 84 | 13206CC | 26033D98 | 008B9794D | 0AC3D699BE | 0 | 0 | 0 | 1 | 1 | 10 | 00 | 11 |
| 324 | 85 | 0640D98 | 4C067B31 | 01172F29B | 1587AD337C | 0 | 0 | 0 | 1 | 1 | 11 | 10 | 00 |
| 325 | 86 | 0C81B30 | 180CF662 | 022E5E537 | 2B0F5A66F9 | 1 | 0 | 0 | 0 | 0 | 00 | 11 | 10 |
| 326 | 87 | 1903660 | 3019ECC5 | 045CBCA6F | 561EB4CDF3 | 1 | 0 | 0 | 0 | 1 | 10 | 00 | 11 |
| 327 | 88 | 1206CC1 | 6033D98A | 08B9794DE | 2C3D699BE6 | 0 | 0 | 1 | 0 | 1 | 11 | 10 | 00 |
| 328 | 89 | 040D983 | 4067B315 | 1172F29BD | 587AD337CC | 0 | 0 | 0 | 0 | 1 | 11 | 11 | 10 |
| 329 | 90 | 081B306 | 00CF662A | 02E5E537A | 30F5A66F98 | 1 | 1 | 0 | 1 | 0 | 10 | 11 | 11 |
| 330 | 91 | 103660C | 019ECC55 | 05CBCA6F4 | 61EB4CDF31 | 0 | 1 | 0 | 1 | 0 | 10 | 10 | 11 |
| 331 | 92 | 006CC19 | 033D98AB | 0B9794DE8 | 43D699BE62 | 0 | 0 | 1 | 1 | 0 | 01 | 10 | 10 |
| 332 | 93 | 00D9833 | 067B3156 | 172F29BD0 | 07AD337CC5 | 0 | 0 | 0 | 1 | 0 | 01 | 01 | 10 |
| 333 | 94 | 01B3066 | 0CF662AC | 0E5E537A0 | 0F5A66F98B | 0 | 1 | 1 | 0 | 1 | 11 | 01 | 01 |
| 334 | 95 | 03660CD | 19ECC559 | 1CBCA6F41 | 1EB4CDF317 | 0 | 1 | 1 | 1 | 0 | 11 | 11 | 01 |
| 335 | 96 | 06CC19B | 33D98AB2 | 19794DE83 | 3D699BE62F | 0 | 1 | 1 | 0 | 1 | 11 | 11 | 11 |
| 336 | 97 | 0D98336 | 67B31565 | 12F29BD06 | 7AD337CC5F | 1 | 1 | 0 | 1 | 0 | 10 | 11 | 11 |
| 337 | 98 | 1B3066D | 4F662ACA | 05E537A0C | 75A66F98BF | 1 | 0 | 0 | 1 | 0 | 10 | 10 | 11 |
| 338 | 99 | 1660CDB | 1ECC5594 | 0BCA6F418 | 6B4CDF317E | 0 | 1 | 1 | 0 | 0 | 01 | 10 | 10 |
| 339 | 100 | 0CC19B7 | 3D98AB29 | 1794DE831 | 5699BE62FC | 1 | 1 | 0 | 1 | 0 | 10 | 01 | 10 |
| 340 | 101 | 198336F | 7B315653 | 0F29BD062 | 2D337CC5F9 | 1 | 0 | 1 | 0 | 0 | 11 | 10 | 01 |

*Sample Data*

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 341 | 102 | 13066DE | 7662ACA7 | 1E537A0C5 | 5A66F98BF2 | 0 | 0 | 1 | 0 | 0 | 00 | 11 | 10 |
| 342 | 103 | 060CDBC | 6CC5594F | 1CA6F418B | 34CDF317E4 | 0 | 1 | 1 | 1 | 1 | 11 | 00 | 11 |
| 343 | 104 | 0C19B78 | 598AB29F | 194DE8317 | 699BE62FC9 | 1 | 1 | 1 | 1 | 1 | 00 | 11 | 00 |
| 344 | 105 | 18336F1 | 3315653F | 129BD062E | 5337CC5F92 | 1 | 0 | 0 | 0 | 1 | 10 | 00 | 11 |
| 345 | 106 | 1066DE2 | 662ACA7E | 0537A0C5C | 266F98BF25 | 0 | 0 | 0 | 0 | 0 | 11 | 10 | 00 |
| 346 | 107 | 00CDBC5 | 4C5594FD | 0A6F418B9 | 4CDF317E4B | 0 | 0 | 1 | 1 | 1 | 00 | 11 | 10 |
| 347 | 108 | 019B78B | 18AB29FA | 14DE83172 | 19BE62FC96 | 0 | 1 | 0 | 1 | 0 | 11 | 00 | 11 |
| 348 | 109 | 0336F16 | 315653F4 | 09BD062E5 | 337CC5F92C | 0 | 0 | 1 | 0 | 0 | 01 | 11 | 00 |
| 349 | 110 | 066DE2D | 62ACA7E8 | 137A0C5CA | 66F98BF258 | 0 | 1 | 0 | 1 | 1 | 10 | 01 | 11 |
| 350 | 111 | 0CDBC5B | 45594FD1 | 06F418B95 | 4DF317E4B1 | 1 | 0 | 0 | 1 | 0 | 11 | 10 | 01 |
| 351 | 112 | 19B78B6 | 0AB29FA2 | 0DE83172B | 1BE62FC962 | 1 | 1 | 1 | 1 | 1 | 01 | 11 | 10 |
| 352 | 113 | 136F16C | 15653F45 | 1BD062E57 | 37CC5F92C5 | 0 | 0 | 1 | 1 | 1 | 10 | 01 | 11 |
| 353 | 114 | 06DE2D9 | 2ACA7E8B | 17A0C5CAE | 6F98BF258B | 0 | 1 | 0 | 1 | 0 | 11 | 10 | 01 |
| 354 | 115 | 0DBC5B2 | 5594FD16 | 0F418B95D | 5F317E4B16 | 1 | 1 | 1 | 0 | 0 | 01 | 11 | 10 |
| 355 | 116 | 1B78B64 | 2B29FA2C | 1E83172BB | 3E62FC962C | 1 | 0 | 1 | 0 | 1 | 10 | 01 | 11 |
| 356 | 117 | 16F16C8 | 5653F458 | 1D062E577 | 7CC5F92C58 | 0 | 0 | 1 | 1 | 0 | 11 | 10 | 01 |
| 357 | 118 | 0DE2D91 | 2CA7E8B0 | 1A0C5CAEF | 798BF258B1 | 1 | 1 | 1 | 1 | 1 | 01 | 11 | 10 |
| 358 | 119 | 1BC5B23 | 594FD161 | 1418B95DF | 7317E4B163 | 1 | 0 | 0 | 0 | 0 | 10 | 01 | 11 |
| 359 | 120 | 178B647 | 329FA2C2 | 083172BBF | 662FC962C7 | 0 | 1 | 1 | 0 | 0 | 11 | 10 | 01 |
| 360 | 121 | 0F16C8E | 653F4584 | 1062E577F | 4C5F92C58E | 1 | 0 | 0 | 0 | 0 | 00 | 11 | 10 |
| 361 | 122 | 1E2D91C | 4A7E8B09 | 00C5CAEFE | 18BF258B1C | 1 | 0 | 0 | 1 | 0 | 11 | 00 | 11 |
| 362 | 123 | 1C5B238 | 14FD1613 | 018B95DFC | 317E4B1639 | 1 | 1 | 0 | 0 | 1 | 01 | 11 | 00 |
| 363 | 124 | 18B6471 | 29FA2C27 | 03172BBF9 | 62FC962C72 | 1 | 1 | 0 | 1 | 0 | 01 | 01 | 11 |
| 364 | 125 | 116C8E2 | 53F4584E | 062E577F3 | 45F92C58E4 | 0 | 1 | 0 | 1 | 1 | 11 | 01 | 01 |

# 2 FREQUENCY HOPPING SAMPLE DATA

The section contains three sets of sample data showing the basic and adapted hopping schemes for different combinations of addresses and initial clock values.

*Sample Data*

## 2.1 FIRST SET

```
Hop sequence {k} for PAGE SCAN/INQUIRY SCAN SUBSTATE:
CLKN start:      0x0000000
UAP / LAP:       0x00000000
#ticks:          0000 | 1000 | 2000 | 3000 | 4000 | 5000 | 6000 | 7000 |
                 ---------------------------------------------------------
0x0000000:          0 |    2 |    4 |    6 |    8 |   10 |   12 |   14 |
0x0008000:         16 |   18 |   20 |   22 |   24 |   26 |   28 |   30 |
0x0010000:         32 |   34 |   36 |   38 |   40 |   42 |   44 |   46 |
0x0018000:         48 |   50 |   52 |   54 |   56 |   58 |   60 |   62 |
0x0020000:          0 |    2 |    4 |    6 |    8 |   10 |   12 |   14 |
0x0028000:         16 |   18 |   20 |   22 |   24 |   26 |   28 |   30 |
0x0030000:         32 |   34 |   36 |   38 |   40 |   42 |   44 |   46 |
0x0038000:         48 |   50 |   52 |   54 |   56 |   58 |   60 |   62 |

Hop sequence {k} for PAGE STATE/INQUIRY SUBSTATE:
CLKE start:      0x0000000
UAP / LAP:       0x00000000
#ticks:          00 01 02 03 | 04 05 06 07 | 08 09 0a 0b | 0c 0d 0e 0f |
                 -----------------------------------------------------------
0x0000000:       48 50 09 13 | 52 54 41 45 | 56 58 11 15 | 60 62 43 47 |
0x0000010:       00 02 64 68 | 04 06 17 21 | 08 10 66 70 | 12 14 19 23 |
0x0000020:       48 50 09 13 | 52 54 41 45 | 56 58 11 15 | 60 62 43 47 |
0x0000030:       00 02 64 68 | 04 06 17 21 | 08 10 66 70 | 12 14 19 23 |
...
0x0001000:       48 18 09 05 | 20 22 33 37 | 24 26 03 07 | 28 30 35 39 |
0x0001010:       32 34 72 76 | 36 38 25 29 | 40 42 74 78 | 44 46 27 31 |
0x0001020:       48 18 09 05 | 20 22 33 37 | 24 26 03 07 | 28 30 35 39 |
0x0001030:       32 34 72 76 | 36 38 25 29 | 40 42 74 78 | 44 46 27 31 |
...
0x0002000:       16 18 01 05 | 52 54 41 45 | 56 58 11 15 | 60 62 43 47 |
0x0002010:       00 02 64 68 | 04 06 17 21 | 08 10 66 70 | 12 14 19 23 |
0x0002020:       16 18 01 05 | 52 54 41 45 | 56 58 11 15 | 60 62 43 47 |
0x0002030:       00 02 64 68 | 04 06 17 21 | 08 10 66 70 | 12 14 19 23 |
...
0x0003000:       48 50 09 13 | 52 22 41 37 | 24 26 03 07 | 28 30 35 39 |
0x0003010:       32 34 72 76 | 36 38 25 29 | 40 42 74 78 | 44 46 27 31 |
0x0003020:       48 50 09 13 | 52 22 41 37 | 24 26 03 07 | 28 30 35 39 |
0x0003030:       32 34 72 76 | 36 38 25 29 | 40 42 74 78 | 44 46 27 31 |

Hop sequence {k} for SLAVE PAGE RESPONSE SUBSTATE:
CLKN* =          0x0000010
UAP / LAP:       0x00000000
#ticks:          00 | 02 04 | 06 08 | 0a 0c | 0e 10 | 12 14 | 16 18 | 1a 1c | 1e
                 ------------------------------------------------------------------
0x0000012:       64 | 02 68 | 04 17 | 06 21 | 08 66 | 10 70 | 12 19 | 14 23 | 16
0x0000032:       01 | 18 05 | 20 33 | 22 37 | 24 03 | 26 07 | 28 35 | 30 39 | 32
0x0000052:       72 | 34 76 | 36 25 | 38 29 | 40 74 | 42 78 | 44 27 | 46 31 | 48
0x0000072:       09 | 50 13 | 52 41 | 54 45 | 56 11 | 58 15 | 60 43 | 62 47 | 00

Hop sequence {k} for MASTER PAGE RESPONSE SUBSTATE:
Offset value:    24
CLKE* =          0x0000012
UAP / LAP:       0x00000000
#ticks:          00 02 | 04 06 | 08 0a | 0c 0e | 10 12 | 14 16 | 18 1a | 1c 1e |
                 ------------------------------------------------------------------
0x0000014:       02 68 | 04 17 | 06 21 | 08 66 | 10 70 | 12 19 | 14 23 | 16 01 |
0x0000034:       18 05 | 20 33 | 22 37 | 24 03 | 26 07 | 28 35 | 30 39 | 32 72 |
0x0000054:       34 76 | 36 25 | 38 29 | 40 74 | 42 78 | 44 27 | 46 31 | 48 09 |
0x0000074:       50 13 | 52 41 | 54 45 | 56 11 | 58 15 | 60 43 | 62 47 | 00 64 |
```

Hop sequence {k} for CONNECTION STATE (Basic channel hopping sequence; ie, non-AFH):
CLK start: 0x0000010
UAP/LAP: 0x00000000

| #ticks: | 00 02 | 04 06 | 08 0a | 0c 0e | 10 12 | 14 16 | 18 1a | 1c 1e |
|---|---|---|---|---|---|---|---|---|
| 0x0000010: | 08 66 | 10 70 | 12 19 | 14 23 | 16 01 | 18 05 | 20 33 | 22 37 |
| 0x0000030: | 24 03 | 26 07 | 28 35 | 30 39 | 32 72 | 34 76 | 36 25 | 38 29 |
| 0x0000050: | 40 74 | 42 78 | 44 27 | 46 31 | 48 09 | 50 13 | 52 41 | 54 45 |
| 0x0000070: | 56 11 | 58 15 | 60 43 | 62 47 | 32 17 | 36 19 | 34 49 | 38 51 |
| 0x0000090: | 40 21 | 44 23 | 42 53 | 46 55 | 48 33 | 52 35 | 50 65 | 54 67 |
| 0x00000b0: | 56 37 | 60 39 | 58 69 | 62 71 | 64 25 | 68 27 | 66 57 | 70 59 |
| 0x00000d0: | 72 29 | 76 31 | 74 61 | 78 63 | 01 41 | 05 43 | 03 73 | 07 75 |
| 0x00000f0: | 09 45 | 13 47 | 11 77 | 15 00 | 64 49 | 66 53 | 68 02 | 70 06 |
| 0x0000110: | 01 51 | 03 55 | 05 04 | 07 08 | 72 57 | 74 61 | 76 10 | 78 14 |
| 0x0000130: | 09 59 | 11 63 | 13 12 | 15 16 | 17 65 | 19 69 | 21 18 | 23 22 |
| 0x0000150: | 33 67 | 35 71 | 37 20 | 39 24 | 25 73 | 27 77 | 29 26 | 31 30 |
| 0x0000170: | 41 75 | 43 00 | 45 28 | 47 32 | 17 02 | 21 04 | 19 34 | 23 36 |
| 0x0000190: | 33 06 | 37 08 | 35 38 | 39 40 | 25 10 | 29 12 | 27 42 | 31 44 |
| 0x00001b0: | 41 14 | 45 16 | 43 46 | 47 48 | 49 18 | 53 20 | 51 50 | 55 52 |
| 0x00001d0: | 65 22 | 69 24 | 67 54 | 71 56 | 57 26 | 61 28 | 59 58 | 63 60 |
| 0x00001f0: | 73 30 | 77 32 | 75 62 | 00 64 | 49 34 | 51 42 | 57 66 | 59 74 |
| 0x0000210: | 53 36 | 55 44 | 61 68 | 63 76 | 65 50 | 67 58 | 73 03 | 75 11 |
| 0x0000230: | 69 52 | 71 60 | 77 05 | 00 13 | 02 38 | 04 46 | 10 70 | 12 78 |
| 0x0000250: | 06 40 | 08 48 | 14 72 | 16 01 | 18 54 | 20 62 | 26 07 | 28 15 |
| 0x0000270: | 22 56 | 24 64 | 30 09 | 32 17 | 02 66 | 06 74 | 10 19 | 14 27 |
| 0x0000290: | 04 70 | 08 78 | 12 23 | 16 31 | 18 03 | 22 11 | 26 35 | 30 43 |
| 0x00002b0: | 20 07 | 24 15 | 28 39 | 32 47 | 34 68 | 38 76 | 42 21 | 46 29 |
| 0x00002d0: | 36 72 | 40 01 | 44 25 | 48 33 | 50 05 | 54 13 | 58 37 | 62 45 |
| 0x00002f0: | 52 09 | 56 17 | 60 41 | 64 49 | 34 19 | 36 35 | 50 51 | 52 67 |
| 0x0000310: | 38 21 | 40 37 | 54 53 | 56 69 | 42 27 | 44 43 | 58 59 | 60 75 |
| 0x0000330: | 46 29 | 48 45 | 62 61 | 64 77 | 66 23 | 68 39 | 03 55 | 05 71 |
| 0x0000350: | 70 25 | 72 41 | 07 57 | 09 73 | 74 31 | 76 47 | 11 63 | 13 00 |
| 0x0000370: | 78 33 | 01 49 | 15 65 | 17 02 | 66 51 | 70 67 | 03 04 | 07 20 |
| 0x0000390: | 68 55 | 72 71 | 05 08 | 09 24 | 74 59 | 78 75 | 11 12 | 15 28 |
| 0x00003b0: | 76 63 | 01 00 | 13 16 | 17 32 | 19 53 | 23 69 | 35 06 | 39 22 |
| 0x00003d0: | 21 57 | 25 73 | 37 10 | 41 26 | 27 61 | 31 77 | 43 14 | 47 30 |
| 0x00003f0: | 29 65 | 33 02 | 45 18 | 49 34 | 19 04 | 21 08 | 23 20 | 25 24 |

*Sample Data*

Hop Sequence {k} for CONNECTION STATE (Adapted channel hopping sequence with all channel used; ie, AFH(79)):

CLK start: 0x0000010

ULAP: 0x00000000

Used Channels:0x7fffffffffffffffff

| #ticks: | 00 02 | 04 06 | 08 0a | 0c 0e | 10 12 | 14 16 | 18 1a | 1c 1e |
|---|---|---|---|---|---|---|---|---|
| 0x0000010 | 08 08 | 10 10 | 12 12 | 14 14 | 16 16 | 18 18 | 20 20 | 22 22 |
| 0x0000030 | 24 24 | 26 26 | 28 28 | 30 30 | 32 32 | 34 34 | 36 36 | 38 38 |
| 0x0000050 | 40 40 | 42 42 | 44 44 | 46 46 | 48 48 | 50 50 | 52 52 | 54 54 |
| 0x0000070 | 56 56 | 58 58 | 60 60 | 62 62 | 32 32 | 36 36 | 34 34 | 38 38 |
| 0x0000090 | 40 40 | 44 44 | 42 42 | 46 46 | 48 48 | 52 52 | 50 50 | 54 54 |
| 0x00000b0 | 56 56 | 60 60 | 58 58 | 62 62 | 64 64 | 68 68 | 66 66 | 70 70 |
| 0x00000d0 | 72 72 | 76 76 | 74 74 | 78 78 | 01 01 | 05 05 | 03 03 | 07 07 |
| 0x00000f0 | 09 09 | 13 13 | 11 11 | 15 15 | 64 64 | 66 66 | 68 68 | 70 70 |
| 0x0000110 | 01 01 | 03 03 | 05 05 | 07 07 | 72 72 | 74 74 | 76 76 | 78 78 |
| 0x0000130 | 09 09 | 11 11 | 13 13 | 15 15 | 17 17 | 19 19 | 21 21 | 23 23 |
| 0x0000150 | 33 33 | 35 35 | 37 37 | 39 39 | 25 25 | 27 27 | 29 29 | 31 31 |
| 0x0000170 | 41 41 | 43 43 | 45 45 | 47 47 | 17 17 | 21 21 | 19 19 | 23 23 |
| 0x0000190 | 33 33 | 37 37 | 35 35 | 39 39 | 25 25 | 29 29 | 27 27 | 31 31 |
| 0x00001b0 | 41 41 | 45 45 | 43 43 | 47 47 | 49 49 | 53 53 | 51 51 | 55 55 |
| 0x00001d0 | 65 65 | 69 69 | 67 67 | 71 71 | 57 57 | 61 61 | 59 59 | 63 63 |
| 0x00001f0 | 73 73 | 77 77 | 75 75 | 00 00 | 49 49 | 51 51 | 57 57 | 59 59 |
| 0x0000210 | 53 53 | 55 55 | 61 61 | 63 63 | 65 65 | 67 67 | 73 73 | 75 75 |
| 0x0000230 | 69 69 | 71 71 | 77 77 | 00 00 | 02 02 | 04 04 | 10 10 | 12 12 |
| 0x0000250 | 06 06 | 08 08 | 14 14 | 16 16 | 18 18 | 20 20 | 26 26 | 28 28 |
| 0x0000270 | 22 22 | 24 24 | 30 30 | 32 32 | 02 02 | 06 06 | 10 10 | 14 14 |
| 0x0000290 | 04 04 | 08 08 | 12 12 | 16 16 | 18 18 | 22 22 | 26 26 | 30 30 |
| 0x00002b0 | 20 20 | 24 24 | 28 28 | 32 32 | 34 34 | 38 38 | 42 42 | 46 46 |
| 0x00002d0 | 36 36 | 40 40 | 44 44 | 48 48 | 50 50 | 54 54 | 58 58 | 62 62 |
| 0x00002f0 | 52 52 | 56 56 | 60 60 | 64 64 | 34 34 | 36 36 | 50 50 | 52 52 |
| 0x0000310 | 38 38 | 40 40 | 54 54 | 56 56 | 42 42 | 44 44 | 58 58 | 60 60 |
| 0x0000330 | 46 46 | 48 48 | 62 62 | 64 64 | 66 66 | 68 68 | 03 03 | 05 05 |
| 0x0000350 | 70 70 | 72 72 | 07 07 | 09 09 | 74 74 | 76 76 | 11 11 | 13 13 |
| 0x0000370 | 78 78 | 01 01 | 15 15 | 17 17 | 66 66 | 70 70 | 03 03 | 07 07 |
| 0x0000390 | 68 68 | 72 72 | 05 05 | 09 09 | 74 74 | 78 78 | 11 11 | 15 15 |
| 0x00003b0 | 76 76 | 01 01 | 13 13 | 17 17 | 19 19 | 23 23 | 35 35 | 39 39 |
| 0x00003d0 | 21 21 | 25 25 | 37 37 | 41 41 | 27 27 | 31 31 | 43 43 | 47 47 |
| 0x00003f0 | 29 29 | 33 33 | 45 45 | 49 49 | 19 19 | 21 21 | 23 23 | 25 25 |

*Sample Data*

Hop Sequence {k} for CONNECTION STATE (Adapted channel hopping sequence with channels 0 to 21 unused):

CLK start: 0x0000010

ULAP: 0x00000000

Used Channels:0x7fffffffffffc00000

| #ticks: | 00 02 | 04 06 | 08 0a | 0c 0e | 10 12 | 14 16 | 18 1a | 1c 1e |
|---|---|---|---|---|---|---|---|---|
| 0x0000010 | 30 30 | 32 32 | 34 34 | 36 36 | 38 38 | 40 40 | 42 42 | 22 22 |
| 0x0000030 | 24 24 | 26 26 | 28 28 | 30 30 | 32 32 | 34 34 | 36 36 | 38 38 |
| 0x0000050 | 40 40 | 42 42 | 44 44 | 46 46 | 48 48 | 50 50 | 52 52 | 54 54 |
| 0x0000070 | 56 56 | 58 58 | 60 60 | 62 62 | 32 32 | 36 36 | 34 34 | 38 38 |
| 0x0000090 | 40 40 | 44 44 | 42 42 | 46 46 | 48 48 | 52 52 | 50 50 | 54 54 |
| 0x00000b0 | 56 56 | 60 60 | 58 58 | 62 62 | 64 64 | 68 68 | 66 66 | 70 70 |
| 0x00000d0 | 72 72 | 76 76 | 74 74 | 78 78 | 45 45 | 49 49 | 47 47 | 51 51 |
| 0x00000f0 | 53 53 | 57 57 | 55 55 | 59 59 | 64 64 | 66 66 | 68 68 | 70 70 |
| 0x0000110 | 45 45 | 47 47 | 49 49 | 51 51 | 72 72 | 74 74 | 76 76 | 78 78 |
| 0x0000130 | 53 53 | 55 55 | 57 57 | 59 59 | 61 61 | 63 63 | 65 65 | 23 23 |
| 0x0000150 | 33 33 | 35 35 | 37 37 | 39 39 | 25 25 | 27 27 | 29 29 | 31 31 |
| 0x0000170 | 41 41 | 43 43 | 45 45 | 47 47 | 61 61 | 65 65 | 63 63 | 23 23 |
| 0x0000190 | 33 33 | 37 37 | 35 35 | 39 39 | 25 25 | 29 29 | 27 27 | 31 31 |
| 0x00001b0 | 41 41 | 45 45 | 43 43 | 47 47 | 49 49 | 53 53 | 51 51 | 55 55 |
| 0x00001d0 | 65 65 | 69 69 | 67 67 | 71 71 | 57 57 | 61 61 | 59 59 | 63 63 |
| 0x00001f0 | 73 73 | 77 77 | 75 75 | 66 66 | 49 49 | 51 51 | 57 57 | 59 59 |
| 0x0000210 | 53 53 | 55 55 | 61 61 | 63 63 | 65 65 | 67 67 | 73 73 | 75 75 |
| 0x0000230 | 69 69 | 71 71 | 77 77 | 66 66 | 68 68 | 70 70 | 76 76 | 78 78 |
| 0x0000250 | 72 72 | 74 74 | 23 23 | 25 25 | 27 27 | 29 29 | 26 26 | 28 28 |
| 0x0000270 | 22 22 | 24 24 | 30 30 | 32 32 | 68 68 | 72 72 | 76 76 | 23 23 |
| 0x0000290 | 70 70 | 74 74 | 78 78 | 25 25 | 27 27 | 22 22 | 26 26 | 30 30 |
| 0x00002b0 | 29 29 | 24 24 | 28 28 | 32 32 | 34 34 | 38 38 | 42 42 | 46 46 |
| 0x00002d0 | 36 36 | 40 40 | 44 44 | 48 48 | 50 50 | 54 54 | 58 58 | 62 62 |
| 0x00002f0 | 52 52 | 56 56 | 60 60 | 64 64 | 34 34 | 36 36 | 50 50 | 52 52 |
| 0x0000310 | 38 38 | 40 40 | 54 54 | 56 56 | 42 42 | 44 44 | 58 58 | 60 60 |
| 0x0000330 | 46 46 | 48 48 | 62 62 | 64 64 | 66 66 | 68 68 | 34 34 | 36 36 |
| 0x0000350 | 70 70 | 72 72 | 38 38 | 40 40 | 74 74 | 76 76 | 42 42 | 44 44 |
| 0x0000370 | 78 78 | 32 32 | 46 46 | 48 48 | 66 66 | 70 70 | 34 34 | 38 38 |
| 0x0000390 | 68 68 | 72 72 | 36 36 | 40 40 | 74 74 | 78 78 | 42 42 | 46 46 |
| 0x00003b0 | 76 76 | 32 32 | 44 44 | 48 48 | 50 50 | 23 23 | 35 35 | 39 39 |
| 0x00003d0 | 52 52 | 25 25 | 37 37 | 41 41 | 27 27 | 31 31 | 43 43 | 47 47 |
| 0x00003f0 | 29 29 | 33 33 | 45 45 | 49 49 | 50 50 | 52 52 | 23 23 | 25 25 |

*Sample Data*

Hop Sequence {k} for CONNECTION STATE (Adapted channel hopping sequence with even channels used):

CLK start: 0x0000010

ULAP: 0x00000000

Used Channels:0x55555555555555555

| #ticks: | 00 02 | 04 06 | 08 0a | 0c 0e | 10 12 | 14 16 | 18 1a | 1c 1e |
|---|---|---|---|---|---|---|---|---|
| 0x0000010 | 08 08 | 10 10 | 12 12 | 14 14 | 16 16 | 18 18 | 20 20 | 22 22 |
| 0x0000030 | 24 24 | 26 26 | 28 28 | 30 30 | 32 32 | 34 34 | 36 36 | 38 38 |
| 0x0000050 | 40 40 | 42 42 | 44 44 | 46 46 | 48 48 | 50 50 | 52 52 | 54 54 |
| 0x0000070 | 56 56 | 58 58 | 60 60 | 62 62 | 32 32 | 36 36 | 34 34 | 38 38 |
| 0x0000090 | 40 40 | 44 44 | 42 42 | 46 46 | 48 48 | 52 52 | 50 50 | 54 54 |
| 0x00000b0 | 56 56 | 60 60 | 58 58 | 62 62 | 64 64 | 68 68 | 66 66 | 70 70 |
| 0x00000d0 | 72 72 | 76 76 | 74 74 | 78 78 | 00 00 | 04 04 | 02 02 | 06 06 |
| 0x00000f0 | 08 08 | 12 12 | 10 10 | 14 14 | 64 64 | 66 66 | 68 68 | 70 70 |
| 0x0000110 | 00 00 | 02 02 | 04 04 | 06 06 | 72 72 | 74 74 | 76 76 | 78 78 |
| 0x0000130 | 08 08 | 10 10 | 12 12 | 14 14 | 16 16 | 18 18 | 20 20 | 22 22 |
| 0x0000150 | 32 32 | 34 34 | 36 36 | 38 38 | 24 24 | 26 26 | 28 28 | 30 30 |
| 0x0000170 | 40 40 | 42 42 | 44 44 | 46 46 | 16 16 | 20 20 | 18 18 | 22 22 |
| 0x0000190 | 32 32 | 36 36 | 34 34 | 38 38 | 24 24 | 28 28 | 26 26 | 30 30 |
| 0x00001b0 | 40 40 | 44 44 | 42 42 | 46 46 | 48 48 | 52 52 | 50 50 | 54 54 |
| 0x00001d0 | 64 64 | 68 68 | 66 66 | 70 70 | 56 56 | 60 60 | 58 58 | 62 62 |
| 0x00001f0 | 72 72 | 76 76 | 74 74 | 00 00 | 48 48 | 50 50 | 56 56 | 58 58 |
| 0x0000210 | 52 52 | 54 54 | 60 60 | 62 62 | 64 64 | 66 66 | 72 72 | 74 74 |
| 0x0000230 | 68 68 | 70 70 | 76 76 | 00 00 | 02 02 | 04 04 | 10 10 | 12 12 |
| 0x0000250 | 06 06 | 08 08 | 14 14 | 16 16 | 18 18 | 20 20 | 26 26 | 28 28 |
| 0x0000270 | 22 22 | 24 24 | 30 30 | 32 32 | 02 02 | 06 06 | 10 10 | 14 14 |
| 0x0000290 | 04 04 | 08 08 | 12 12 | 16 16 | 18 18 | 22 22 | 26 26 | 30 30 |
| 0x00002b0 | 20 20 | 24 24 | 28 28 | 32 32 | 34 34 | 38 38 | 42 42 | 46 46 |
| 0x00002d0 | 36 36 | 40 40 | 44 44 | 48 48 | 50 50 | 54 54 | 58 58 | 62 62 |
| 0x00002f0 | 52 52 | 56 56 | 60 60 | 64 64 | 34 34 | 36 36 | 50 50 | 52 52 |
| 0x0000310 | 38 38 | 40 40 | 54 54 | 56 56 | 42 42 | 44 44 | 58 58 | 60 60 |
| 0x0000330 | 46 46 | 48 48 | 62 62 | 64 64 | 66 66 | 68 68 | 00 00 | 02 02 |
| 0x0000350 | 70 70 | 72 72 | 04 04 | 06 06 | 74 74 | 76 76 | 08 08 | 10 10 |
| 0x0000370 | 78 78 | 78 78 | 12 12 | 14 14 | 66 66 | 70 70 | 00 00 | 04 04 |
| 0x0000390 | 68 68 | 72 72 | 02 02 | 06 06 | 74 74 | 78 78 | 08 08 | 12 12 |
| 0x00003b0 | 76 76 | 78 78 | 10 10 | 14 14 | 16 16 | 20 20 | 32 32 | 36 36 |
| 0x00003d0 | 18 18 | 22 22 | 34 34 | 38 38 | 24 24 | 28 28 | 40 40 | 44 44 |
| 0x00003f0 | 26 26 | 30 30 | 42 42 | 46 46 | 16 16 | 18 18 | 20 20 | 22 22 |

*Sample Data*

Hop Sequence {k} for CONNECTION STATE (Adapted channel hopping sequence with odd channels used):

CLK start: 0x0000010

ULAP: 0x00000000

Used Channels:0x2aaaaaaaaaaaaaaaaaaa

| #ticks: | 00 02 | 04 06 | 08 0a | 0c 0e | 10 12 | 14 16 | 18 1a | 1c 1e |
|---|---|---|---|---|---|---|---|---|
| 0x0000010 | 09 09 | 11 11 | 13 13 | 15 15 | 17 17 | 19 19 | 21 21 | 23 23 |
| 0x0000030 | 25 25 | 27 27 | 29 29 | 31 31 | 33 33 | 35 35 | 37 37 | 39 39 |
| 0x0000050 | 41 41 | 43 43 | 45 45 | 47 47 | 49 49 | 51 51 | 53 53 | 55 55 |
| 0x0000070 | 57 57 | 59 59 | 61 61 | 63 63 | 33 33 | 37 37 | 35 35 | 39 39 |
| 0x0000090 | 41 41 | 45 45 | 43 43 | 47 47 | 49 49 | 53 53 | 51 51 | 55 55 |
| 0x00000b0 | 57 57 | 61 61 | 59 59 | 63 63 | 65 65 | 69 69 | 67 67 | 71 71 |
| 0x00000d0 | 73 73 | 77 77 | 75 75 | 01 01 | 01 01 | 05 05 | 03 03 | 07 07 |
| 0x00000f0 | 09 09 | 13 13 | 11 11 | 15 15 | 65 65 | 67 67 | 69 69 | 71 71 |
| 0x0000110 | 01 01 | 03 03 | 05 05 | 07 07 | 73 73 | 75 75 | 77 77 | 01 01 |
| 0x0000130 | 09 09 | 11 11 | 13 13 | 15 15 | 17 17 | 19 19 | 21 21 | 23 23 |
| 0x0000150 | 33 33 | 35 35 | 37 37 | 39 39 | 25 25 | 27 27 | 29 29 | 31 31 |
| 0x0000170 | 41 41 | 43 43 | 45 45 | 47 47 | 17 17 | 21 21 | 19 19 | 23 23 |
| 0x0000190 | 33 33 | 37 37 | 35 35 | 39 39 | 25 25 | 29 29 | 27 27 | 31 31 |
| 0x00001b0 | 41 41 | 45 45 | 43 43 | 47 47 | 49 49 | 53 53 | 51 51 | 55 55 |
| 0x00001d0 | 65 65 | 69 69 | 67 67 | 71 71 | 57 57 | 61 61 | 59 59 | 63 63 |
| 0x00001f0 | 73 73 | 77 77 | 75 75 | 03 03 | 49 49 | 51 51 | 57 57 | 59 59 |
| 0x0000210 | 53 53 | 55 55 | 61 61 | 63 63 | 65 65 | 67 67 | 73 73 | 75 75 |
| 0x0000230 | 69 69 | 71 71 | 77 77 | 03 03 | 05 05 | 07 07 | 13 13 | 15 15 |
| 0x0000250 | 09 09 | 11 11 | 17 17 | 19 19 | 21 21 | 23 23 | 29 29 | 31 31 |
| 0x0000270 | 25 25 | 27 27 | 33 33 | 35 35 | 05 05 | 09 09 | 13 13 | 17 17 |
| 0x0000290 | 07 07 | 11 11 | 15 15 | 19 19 | 21 21 | 25 25 | 29 29 | 33 33 |
| 0x00002b0 | 23 23 | 27 27 | 31 31 | 35 35 | 37 37 | 41 41 | 45 45 | 49 49 |
| 0x00002d0 | 39 39 | 43 43 | 47 47 | 51 51 | 53 53 | 57 57 | 61 61 | 65 65 |
| 0x00002f0 | 55 55 | 59 59 | 63 63 | 67 67 | 37 37 | 39 39 | 53 53 | 55 55 |
| 0x0000310 | 41 41 | 43 43 | 57 57 | 59 59 | 45 45 | 47 47 | 61 61 | 63 63 |
| 0x0000330 | 49 49 | 51 51 | 65 65 | 67 67 | 69 69 | 71 71 | 03 03 | 05 05 |
| 0x0000350 | 73 73 | 75 75 | 07 07 | 09 09 | 77 77 | 01 01 | 11 11 | 13 13 |
| 0x0000370 | 03 03 | 01 01 | 15 15 | 17 17 | 69 69 | 73 73 | 03 03 | 07 07 |
| 0x0000390 | 71 71 | 75 75 | 05 05 | 09 09 | 77 77 | 03 03 | 11 11 | 15 15 |
| 0x00003b0 | 01 01 | 01 01 | 13 13 | 17 17 | 19 19 | 23 23 | 35 35 | 39 39 |
| 0x00003d0 | 21 21 | 25 25 | 37 37 | 41 41 | 27 27 | 31 31 | 43 43 | 47 47 |
| 0x00003f0 | 29 29 | 33 33 | 45 45 | 49 49 | 19 19 | 21 21 | 23 23 | 25 25 |

## 2.2 SECOND SET

```
Hop sequence {k} for PAGE SCAN/INQUIRY SCAN SUBSTATE:
CLKN start:     0x0000000
ULAP:           0x2a96ef25
#ticks:         0000 | 1000 | 2000 | 3000 | 4000 | 5000 | 6000 | 7000 |
                ---------------------------------------------------------
0x0000000:        49 |   13 |   17 |   51 |   55 |   19 |   23 |   53 |
0x0008000:        57 |   21 |   25 |   27 |   31 |   74 |   78 |   29 |
0x0010000:        33 |   76 |    1 |   35 |   39 |    3 |    7 |   37 |
0x0018000:        41 |    5 |    9 |   43 |   47 |   11 |   15 |   45 |
0x0020000:        49 |   13 |   17 |   51 |   55 |   19 |   23 |   53 |
0x0028000:        57 |   21 |   25 |   27 |   31 |   74 |   78 |   29 |
0x0030000:        33 |   76 |    1 |   35 |   39 |    3 |    7 |   37 |
0x0038000:        41 |    5 |    9 |   43 |   47 |   11 |   15 |   45 |

Hop sequence {k} for PAGE STATE/INQUIRY SUBSTATE:
CLKE start:     0x0000000
ULAP:           0x2a96ef25
#ticks:         00 01 02 03 | 04 05 06 07 | 08 09 0a 0b | 0c 0d 0e 0f |
                -----------------------------------------------------------
0x0000000:      41 05 10 04 | 09 43 06 16 | 47 11 18 12 | 15 45 14 32 |
0x0000010:      49 13 34 28 | 17 51 30 24 | 55 19 26 20 | 23 53 22 40 |
0x0000020:      41 05 10 04 | 09 43 06 16 | 47 11 18 12 | 15 45 14 32 |
0x0000030:      49 13 34 28 | 17 51 30 24 | 55 19 26 20 | 23 53 22 40 |
...
0x0001000:      41 21 10 36 | 25 27 38 63 | 31 74 65 59 | 78 29 61 00 |
0x0001010:      33 76 02 75 | 01 35 77 71 | 39 03 73 67 | 07 37 69 08 |
0x0001020:      41 21 10 36 | 25 27 38 63 | 31 74 65 59 | 78 29 61 00 |
0x0001030:      33 76 02 75 | 01 35 77 71 | 39 03 73 67 | 07 37 69 08 |
...
0x0002000:      57 21 42 36 | 09 43 06 16 | 47 11 18 12 | 15 45 14 32 |
0x0002010:      49 13 34 28 | 17 51 30 24 | 55 19 26 20 | 23 53 22 40 |
0x0002020:      57 21 42 36 | 09 43 06 16 | 47 11 18 12 | 15 45 14 32 |
0x0002030:      49 13 34 28 | 17 51 30 24 | 55 19 26 20 | 23 53 22 40 |
...
0x0003000:      41 05 10 04 | 09 27 06 63 | 31 74 65 59 | 78 29 61 00 |
0x0003010:      33 76 02 75 | 01 35 77 71 | 39 03 73 67 | 07 37 69 08 |
0x0003020:      41 05 10 04 | 09 27 06 63 | 31 74 65 59 | 78 29 61 00 |
0x0003030:      33 76 02 75 | 01 35 77 71 | 39 03 73 67 | 07 37 69 08 |

Hop sequence {k} for SLAVE PAGE RESPONSE SUBSTATE:
CLKN* =         0x0000010
ULAP:           0x2a96ef25
#ticks:         00 | 02 04 | 06 08 | 0a 0c | 0e 10 | 12 14 | 16 18 | 1a
                -----------------------------------------------------------
0x0000012:      34 | 13 28 | 17 30 | 51 24 | 55 26 | 19 20 | 23 22 | 53
0x0000032:      42 | 21 36 | 25 38 | 27 63 | 31 65 | 74 59 | 78 61 | 29
0x0000052:      02 | 76 75 | 01 77 | 35 71 | 39 73 | 03 67 | 07 69 | 37
0x0000072:      10 | 05 04 | 09 06 | 43 16 | 47 18 | 11 12 | 15 14 | 45

Hop sequence {k} for MASTER PAGE RESPONSE SUBSTATE:
Offset value:   24
CLKE* =         0x0000012
ULAP:           0x2a96ef25
#ticks:         00 02 | 04 06 | 08 0a | 0c 0e | 10 12 | 14 16 | 18 1a |
                -----------------------------------------------------------
0x0000014:      13 28 | 17 30 | 51 24 | 55 26 | 19 20 | 23 22 | 53 40 |
0x0000034:      21 36 | 25 38 | 27 63 | 31 65 | 74 59 | 78 61 | 29 00 |
0x0000054:      76 75 | 01 77 | 35 71 | 39 73 | 03 67 | 07 69 | 37 08 |
0x0000074:      05 04 | 09 06 | 43 16 | 47 18 | 11 12 | 15 14 | 45 32 |
```

Hop sequence {k} for CONNECTION STATE (Basic channel hopping sequence; ie, non-AFH):
CLK start: 0x0000010
ULAP: 0x2a96ef25

| #ticks: | 00 02 | 04 06 | 08 0a | 0c 0e | 10 12 | 14 16 | 18 1a | 1c 1e |
|---|---|---|---|---|---|---|---|---|
| 0x0000010: | 55 26 | 19 20 | 23 22 | 53 40 | 57 42 | 21 36 | 25 38 | 27 63 |
| 0x0000030: | 31 65 | 74 59 | 78 61 | 29 00 | 33 02 | 76 75 | 01 77 | 35 71 |
| 0x0000050: | 39 73 | 03 67 | 07 69 | 37 08 | 41 10 | 05 04 | 09 06 | 43 16 |
| 0x0000070: | 47 18 | 11 12 | 15 14 | 45 32 | 02 66 | 47 60 | 49 64 | 04 54 |
| 0x0000090: | 06 58 | 51 52 | 53 56 | 08 70 | 10 74 | 55 68 | 57 72 | 59 14 |
| 0x00000b0: | 61 18 | 27 12 | 29 16 | 63 30 | 65 34 | 31 28 | 33 32 | 67 22 |
| 0x00000d0: | 69 26 | 35 20 | 37 24 | 71 38 | 73 42 | 39 36 | 41 40 | 75 46 |
| 0x00000f0: | 77 50 | 43 44 | 45 48 | 00 62 | 26 11 | 69 05 | 73 07 | 36 17 |
| 0x0000110: | 40 19 | 04 13 | 08 15 | 38 25 | 42 27 | 06 21 | 10 23 | 12 48 |
| 0x0000130: | 16 50 | 59 44 | 63 46 | 14 56 | 18 58 | 61 52 | 65 54 | 28 64 |
| 0x0000150: | 32 66 | 75 60 | 00 62 | 30 72 | 34 74 | 77 68 | 02 70 | 20 01 |
| 0x0000170: | 24 03 | 67 76 | 71 78 | 22 09 | 58 43 | 24 37 | 26 41 | 68 47 |
| 0x0000190: | 70 51 | 36 45 | 38 49 | 72 55 | 74 59 | 40 53 | 42 57 | 44 78 |
| 0x00001b0: | 46 03 | 12 76 | 14 01 | 48 07 | 50 11 | 16 05 | 18 09 | 60 15 |
| 0x00001d0: | 62 19 | 28 13 | 30 17 | 64 23 | 66 27 | 32 21 | 34 25 | 52 31 |
| 0x00001f0: | 54 35 | 20 29 | 22 33 | 56 39 | 19 04 | 62 63 | 66 00 | 07 73 |
| 0x0000210: | 11 10 | 54 69 | 58 06 | 23 75 | 27 12 | 70 71 | 74 08 | 76 33 |
| 0x0000230: | 01 49 | 44 29 | 48 45 | 13 35 | 17 51 | 60 31 | 64 47 | 05 41 |
| 0x0000250: | 09 57 | 52 37 | 56 53 | 21 43 | 25 59 | 68 39 | 72 55 | 78 65 |
| 0x0000270: | 03 02 | 46 61 | 50 77 | 15 67 | 51 36 | 17 18 | 19 34 | 41 24 |
| 0x0000290: | 43 40 | 09 22 | 11 38 | 57 28 | 59 44 | 25 26 | 27 42 | 29 63 |
| 0x00002b0: | 31 00 | 76 61 | 78 77 | 45 67 | 47 04 | 13 65 | 15 02 | 37 71 |
| 0x00002d0: | 39 08 | 05 69 | 07 06 | 53 75 | 55 12 | 21 73 | 23 10 | 33 16 |
| 0x00002f0: | 35 32 | 01 14 | 03 30 | 49 20 | 75 60 | 39 48 | 43 56 | 00 66 |
| 0x0000310: | 04 74 | 47 62 | 51 70 | 08 68 | 12 76 | 55 64 | 59 72 | 61 18 |
| 0x0000330: | 65 26 | 29 14 | 33 22 | 69 20 | 73 28 | 37 16 | 41 24 | 77 34 |
| 0x0000350: | 02 42 | 45 30 | 49 38 | 06 36 | 10 44 | 53 32 | 57 40 | 63 50 |
| 0x0000370: | 67 58 | 31 46 | 35 54 | 71 52 | 28 13 | 73 03 | 75 11 | 34 17 |
| 0x0000390: | 36 25 | 02 15 | 04 23 | 42 21 | 44 29 | 10 19 | 12 27 | 14 48 |
| 0x00003b0: | 16 56 | 61 46 | 63 54 | 22 52 | 24 60 | 69 50 | 71 58 | 30 64 |
| 0x00003d0: | 32 72 | 77 62 | 00 70 | 38 68 | 40 76 | 06 66 | 08 74 | 18 01 |
| 0x00003f0: | 20 09 | 65 78 | 67 07 | 26 05 | 44 29 | 32 23 | 36 25 | 70 43 |

*Sample Data*

Hop Sequence {k} for CONNECTION STATE (Adapted channel hopping sequence with all channel used; ie, AFH(79)):

CLK start: 0x0000010

ULAP: 0x2a96ef25

Used Channels:0x7fffffffffffffffff

| #ticks: | 00 02 | 04 06 | 08 0a | 0c 0e | 10 12 | 14 16 | 18 1a | 1c 1e |
|---|---|---|---|---|---|---|---|---|
| 0x0000010 | 55 55 | 19 19 | 23 23 | 53 53 | 57 57 | 21 21 | 25 25 | 27 27 |
| 0x0000030 | 31 31 | 74 74 | 78 78 | 29 29 | 33 33 | 76 76 | 01 01 | 35 35 |
| 0x0000050 | 39 39 | 03 03 | 07 07 | 37 37 | 41 41 | 05 05 | 09 09 | 43 43 |
| 0x0000070 | 47 47 | 11 11 | 15 15 | 45 45 | 02 02 | 47 47 | 49 49 | 04 04 |
| 0x0000090 | 06 06 | 51 51 | 53 53 | 08 08 | 10 10 | 55 55 | 57 57 | 59 59 |
| 0x00000b0 | 61 61 | 27 27 | 29 29 | 63 63 | 65 65 | 31 31 | 33 33 | 67 67 |
| 0x00000d0 | 69 69 | 35 35 | 37 37 | 71 71 | 73 73 | 39 39 | 41 41 | 75 75 |
| 0x00000f0 | 77 77 | 43 43 | 45 45 | 00 00 | 26 26 | 69 69 | 73 73 | 36 36 |
| 0x0000110 | 40 40 | 04 04 | 08 08 | 38 38 | 42 42 | 06 06 | 10 10 | 12 12 |
| 0x0000130 | 16 16 | 59 59 | 63 63 | 14 14 | 18 18 | 61 61 | 65 65 | 28 28 |
| 0x0000150 | 32 32 | 75 75 | 00 00 | 30 30 | 34 34 | 77 77 | 02 02 | 20 20 |
| 0x0000170 | 24 24 | 67 67 | 71 71 | 22 22 | 58 58 | 24 24 | 26 26 | 68 68 |
| 0x0000190 | 70 70 | 36 36 | 38 38 | 72 72 | 74 74 | 40 40 | 42 42 | 44 44 |
| 0x00001b0 | 46 46 | 12 12 | 14 14 | 48 48 | 50 50 | 16 16 | 18 18 | 60 60 |
| 0x00001d0 | 62 62 | 28 28 | 30 30 | 64 64 | 66 66 | 32 32 | 34 34 | 52 52 |
| 0x00001f0 | 54 54 | 20 20 | 22 22 | 56 56 | 19 19 | 62 62 | 66 66 | 07 07 |
| 0x0000210 | 11 11 | 54 54 | 58 58 | 23 23 | 27 27 | 70 70 | 74 74 | 76 76 |
| 0x0000230 | 01 01 | 44 44 | 48 48 | 13 13 | 17 17 | 60 60 | 64 64 | 05 05 |
| 0x0000250 | 09 09 | 52 52 | 56 56 | 21 21 | 25 25 | 68 68 | 72 72 | 78 78 |
| 0x0000270 | 03 03 | 46 46 | 50 50 | 15 15 | 51 51 | 17 17 | 19 19 | 41 41 |
| 0x0000290 | 43 43 | 09 09 | 11 11 | 57 57 | 59 59 | 25 25 | 27 27 | 29 29 |
| 0x00002b0 | 31 31 | 76 76 | 78 78 | 45 45 | 47 47 | 13 13 | 15 15 | 37 37 |
| 0x00002d0 | 39 39 | 05 05 | 07 07 | 53 53 | 55 55 | 21 21 | 23 23 | 33 33 |
| 0x00002f0 | 35 35 | 01 01 | 03 03 | 49 49 | 75 75 | 39 39 | 43 43 | 00 00 |
| 0x0000310 | 04 04 | 47 47 | 51 51 | 08 08 | 12 12 | 55 55 | 59 59 | 61 61 |
| 0x0000330 | 65 65 | 29 29 | 33 33 | 69 69 | 73 73 | 37 37 | 41 41 | 77 77 |
| 0x0000350 | 02 02 | 45 45 | 49 49 | 06 06 | 10 10 | 53 53 | 57 57 | 63 63 |
| 0x0000370 | 67 67 | 31 31 | 35 35 | 71 71 | 28 28 | 73 73 | 75 75 | 34 34 |
| 0x0000390 | 36 36 | 02 02 | 04 04 | 42 42 | 44 44 | 10 10 | 12 12 | 14 14 |
| 0x00003b0 | 16 16 | 61 61 | 63 63 | 22 22 | 24 24 | 69 69 | 71 71 | 30 30 |
| 0x00003d0 | 32 32 | 77 77 | 00 00 | 38 38 | 40 40 | 06 06 | 08 08 | 18 18 |
| 0x00003f0 | 20 20 | 65 65 | 67 67 | 26 26 | 44 44 | 32 32 | 36 36 | 70 70 |

*Sample Data*

Hop Sequence {k} for CONNECTION STATE (Adapted channel hopping sequence with channels 0 to 21 unused):

CLK start: 0x0000010

ULAP: 0x2a96ef25

Used Channels:0x7fffffffffffc00000

| #ticks: | 00 02 | 04 06 | 08 0a | 0c 0e | 10 12 | 14 16 | 18 1a | 1c 1e |
|---|---|---|---|---|---|---|---|---|
| 0x0000010 | 55 55 | 50 50 | 23 23 | 53 53 | 57 57 | 52 52 | 25 25 | 27 27 |
| 0x0000030 | 31 31 | 74 74 | 78 78 | 29 29 | 33 33 | 76 76 | 32 32 | 35 35 |
| 0x0000050 | 39 39 | 34 34 | 38 38 | 37 37 | 41 41 | 36 36 | 40 40 | 43 43 |
| 0x0000070 | 47 47 | 42 42 | 46 46 | 45 45 | 55 55 | 47 47 | 49 49 | 57 57 |
| 0x0000090 | 59 59 | 51 51 | 53 53 | 61 61 | 63 63 | 55 55 | 57 57 | 59 59 |
| 0x00000b0 | 61 61 | 27 27 | 29 29 | 63 63 | 65 65 | 31 31 | 33 33 | 67 67 |
| 0x00000d0 | 69 69 | 35 35 | 37 37 | 71 71 | 73 73 | 39 39 | 41 41 | 75 75 |
| 0x00000f0 | 77 77 | 43 43 | 45 45 | 53 53 | 26 26 | 69 69 | 73 73 | 36 36 |
| 0x0000110 | 40 40 | 57 57 | 61 61 | 38 38 | 42 42 | 59 59 | 63 63 | 65 65 |
| 0x0000130 | 69 69 | 59 59 | 63 63 | 67 67 | 71 71 | 61 61 | 65 65 | 28 28 |
| 0x0000150 | 32 32 | 75 75 | 53 53 | 30 30 | 34 34 | 77 77 | 55 55 | 73 73 |
| 0x0000170 | 24 24 | 67 67 | 71 71 | 22 22 | 58 58 | 24 24 | 26 26 | 68 68 |
| 0x0000190 | 70 70 | 36 36 | 38 38 | 72 72 | 74 74 | 40 40 | 42 42 | 44 44 |
| 0x00001b0 | 46 46 | 65 65 | 67 67 | 48 48 | 50 50 | 69 69 | 71 71 | 60 60 |
| 0x00001d0 | 62 62 | 28 28 | 30 30 | 64 64 | 66 66 | 32 32 | 34 34 | 52 52 |
| 0x00001f0 | 54 54 | 73 73 | 22 22 | 56 56 | 37 37 | 62 62 | 66 66 | 25 25 |
| 0x0000210 | 29 29 | 54 54 | 58 58 | 23 23 | 27 27 | 70 70 | 74 74 | 76 76 |
| 0x0000230 | 76 76 | 44 44 | 48 48 | 31 31 | 35 35 | 60 60 | 64 64 | 23 23 |
| 0x0000250 | 27 27 | 52 52 | 56 56 | 39 39 | 25 25 | 68 68 | 72 72 | 78 78 |
| 0x0000270 | 78 78 | 46 46 | 50 50 | 33 33 | 51 51 | 35 35 | 37 37 | 41 41 |
| 0x0000290 | 43 43 | 27 27 | 29 29 | 57 57 | 59 59 | 25 25 | 27 27 | 29 29 |
| 0x00002b0 | 31 31 | 76 76 | 78 78 | 45 45 | 47 47 | 31 31 | 33 33 | 37 37 |
| 0x00002d0 | 39 39 | 23 23 | 25 25 | 53 53 | 55 55 | 39 39 | 23 23 | 33 33 |
| 0x00002f0 | 35 35 | 76 76 | 78 78 | 49 49 | 75 75 | 39 39 | 43 43 | 40 40 |
| 0x0000310 | 44 44 | 47 47 | 51 51 | 48 48 | 52 52 | 55 55 | 59 59 | 61 61 |
| 0x0000330 | 65 65 | 29 29 | 33 33 | 69 69 | 73 73 | 37 37 | 41 41 | 77 77 |
| 0x0000350 | 42 42 | 45 45 | 49 49 | 46 46 | 50 50 | 53 53 | 57 57 | 63 63 |
| 0x0000370 | 67 67 | 31 31 | 35 35 | 71 71 | 28 28 | 73 73 | 75 75 | 34 34 |
| 0x0000390 | 36 36 | 42 42 | 44 44 | 42 42 | 44 44 | 50 50 | 52 52 | 54 54 |
| 0x00003b0 | 56 56 | 61 61 | 63 63 | 22 22 | 24 24 | 69 69 | 71 71 | 30 30 |
| 0x00003d0 | 32 32 | 77 77 | 40 40 | 38 38 | 40 40 | 46 46 | 48 48 | 58 58 |
| 0x00003f0 | 60 60 | 65 65 | 67 67 | 26 26 | 44 44 | 32 32 | 36 36 | 70 70 |

*Sample Data*

Hop Sequence {k} for CONNECTION STATE (Adapted channel hopping sequence with even channels used):

CLK start: 0x0000010

ULAP: 0x2a96ef25

Used Channels:0x55555555555555555

| #ticks: | 00 02 | 04 06 | 08 0a | 0c 0e | 10 12 | 14 16 | 18 1a | 1c 1e |
|---|---|---|---|---|---|---|---|---|
| 0x0000010 | 52 52 | 16 16 | 20 20 | 50 50 | 54 54 | 18 18 | 22 22 | 24 24 |
| 0x0000030 | 28 28 | 74 74 | 78 78 | 26 26 | 30 30 | 76 76 | 78 78 | 32 32 |
| 0x0000050 | 36 36 | 00 00 | 04 04 | 34 34 | 38 38 | 02 02 | 06 06 | 40 40 |
| 0x0000070 | 44 44 | 08 08 | 12 12 | 42 42 | 02 02 | 44 44 | 46 46 | 04 04 |
| 0x0000090 | 06 06 | 48 48 | 50 50 | 08 08 | 10 10 | 52 52 | 54 54 | 56 56 |
| 0x00000b0 | 58 58 | 24 24 | 26 26 | 60 60 | 62 62 | 28 28 | 30 30 | 64 64 |
| 0x00000d0 | 66 66 | 32 32 | 34 34 | 68 68 | 70 70 | 36 36 | 38 38 | 72 72 |
| 0x00000f0 | 74 74 | 40 40 | 42 42 | 00 00 | 26 26 | 66 66 | 70 70 | 36 36 |
| 0x0000110 | 40 40 | 04 04 | 08 08 | 38 38 | 42 42 | 06 06 | 10 10 | 12 12 |
| 0x0000130 | 16 16 | 56 56 | 60 60 | 14 14 | 18 18 | 58 58 | 62 62 | 28 28 |
| 0x0000150 | 32 32 | 72 72 | 00 00 | 30 30 | 34 34 | 74 74 | 02 02 | 20 20 |
| 0x0000170 | 24 24 | 64 64 | 68 68 | 22 22 | 58 58 | 24 24 | 26 26 | 68 68 |
| 0x0000190 | 70 70 | 36 36 | 38 38 | 72 72 | 74 74 | 40 40 | 42 42 | 44 44 |
| 0x00001b0 | 46 46 | 12 12 | 14 14 | 48 48 | 50 50 | 16 16 | 18 18 | 60 60 |
| 0x00001d0 | 62 62 | 28 28 | 30 30 | 64 64 | 66 66 | 32 32 | 34 34 | 52 52 |
| 0x00001f0 | 54 54 | 20 20 | 22 22 | 56 56 | 14 14 | 62 62 | 66 66 | 02 02 |
| 0x0000210 | 06 06 | 54 54 | 58 58 | 18 18 | 22 22 | 70 70 | 74 74 | 76 76 |
| 0x0000230 | 76 76 | 44 44 | 48 48 | 08 08 | 12 12 | 60 60 | 64 64 | 00 00 |
| 0x0000250 | 04 04 | 52 52 | 56 56 | 16 16 | 20 20 | 68 68 | 72 72 | 78 78 |
| 0x0000270 | 78 78 | 46 46 | 50 50 | 10 10 | 46 46 | 12 12 | 14 14 | 36 36 |
| 0x0000290 | 38 38 | 04 04 | 06 06 | 52 52 | 54 54 | 20 20 | 22 22 | 24 24 |
| 0x00002b0 | 26 26 | 76 76 | 78 78 | 40 40 | 42 42 | 08 08 | 10 10 | 32 32 |
| 0x00002d0 | 34 34 | 00 00 | 02 02 | 48 48 | 50 50 | 16 16 | 18 18 | 28 28 |
| 0x00002f0 | 30 30 | 76 76 | 78 78 | 44 44 | 70 70 | 34 34 | 38 38 | 00 00 |
| 0x0000310 | 04 04 | 42 42 | 46 46 | 08 08 | 12 12 | 50 50 | 54 54 | 56 56 |
| 0x0000330 | 60 60 | 24 24 | 28 28 | 64 64 | 68 68 | 32 32 | 36 36 | 72 72 |
| 0x0000350 | 02 02 | 40 40 | 44 44 | 06 06 | 10 10 | 48 48 | 52 52 | 58 58 |
| 0x0000370 | 62 62 | 26 26 | 30 30 | 66 66 | 28 28 | 68 68 | 70 70 | 34 34 |
| 0x0000390 | 36 36 | 02 02 | 04 04 | 42 42 | 44 44 | 10 10 | 12 12 | 14 14 |
| 0x00003b0 | 16 16 | 56 56 | 58 58 | 22 22 | 24 24 | 64 64 | 66 66 | 30 30 |
| 0x00003d0 | 32 32 | 72 72 | 00 00 | 38 38 | 40 40 | 06 06 | 08 08 | 18 18 |
| 0x00003f0 | 20 20 | 60 60 | 62 62 | 26 26 | 44 44 | 32 32 | 36 36 | 70 70 |

*Sample Data*

Hop Sequence {k} for CONNECTION STATE (Adapted channel hopping sequence with odd channels used):

CLK start: 0x0000010

ULAP: 0x2a96ef25

Used Channels:0x2aaaaaaaaaaaaaaaaaaa

| #ticks: | 00 02 | 04 06 | 08 0a | 0c 0e | 10 12 | 14 16 | 18 1a | 1c 1e |
|---|---|---|---|---|---|---|---|---|
| 0x0000010 | 55 55 | 19 19 | 23 23 | 53 53 | 57 57 | 21 21 | 25 25 | 27 27 |
| 0x0000030 | 31 31 | 77 77 | 03 03 | 29 29 | 33 33 | 01 01 | 01 01 | 35 35 |
| 0x0000050 | 39 39 | 03 03 | 07 07 | 37 37 | 41 41 | 05 05 | 09 09 | 43 43 |
| 0x0000070 | 47 47 | 11 11 | 15 15 | 45 45 | 07 07 | 47 47 | 49 49 | 09 09 |
| 0x0000090 | 11 11 | 51 51 | 53 53 | 13 13 | 15 15 | 55 55 | 57 57 | 59 59 |
| 0x00000b0 | 61 61 | 27 27 | 29 29 | 63 63 | 65 65 | 31 31 | 33 33 | 67 67 |
| 0x00000d0 | 69 69 | 35 35 | 37 37 | 71 71 | 73 73 | 39 39 | 41 41 | 75 75 |
| 0x00000f0 | 77 77 | 43 43 | 45 45 | 05 05 | 31 31 | 69 69 | 73 73 | 41 41 |
| 0x0000110 | 45 45 | 09 09 | 13 13 | 43 43 | 47 47 | 11 11 | 15 15 | 17 17 |
| 0x0000130 | 21 21 | 59 59 | 63 63 | 19 19 | 23 23 | 61 61 | 65 65 | 33 33 |
| 0x0000150 | 37 37 | 75 75 | 05 05 | 35 35 | 39 39 | 77 77 | 07 07 | 25 25 |
| 0x0000170 | 29 29 | 67 67 | 71 71 | 27 27 | 63 63 | 29 29 | 31 31 | 73 73 |
| 0x0000190 | 75 75 | 41 41 | 43 43 | 77 77 | 01 01 | 45 45 | 47 47 | 49 49 |
| 0x00001b0 | 51 51 | 17 17 | 19 19 | 53 53 | 55 55 | 21 21 | 23 23 | 65 65 |
| 0x00001d0 | 67 67 | 33 33 | 35 35 | 69 69 | 71 71 | 37 37 | 39 39 | 57 57 |
| 0x00001f0 | 59 59 | 25 25 | 27 27 | 61 61 | 19 19 | 67 67 | 71 71 | 07 07 |
| 0x0000210 | 11 11 | 59 59 | 63 63 | 23 23 | 27 27 | 75 75 | 01 01 | 03 03 |
| 0x0000230 | 01 01 | 49 49 | 53 53 | 13 13 | 17 17 | 65 65 | 69 69 | 05 05 |
| 0x0000250 | 09 09 | 57 57 | 61 61 | 21 21 | 25 25 | 73 73 | 77 77 | 05 05 |
| 0x0000270 | 03 03 | 51 51 | 55 55 | 15 15 | 51 51 | 17 17 | 19 19 | 41 41 |
| 0x0000290 | 43 43 | 09 09 | 11 11 | 57 57 | 59 59 | 25 25 | 27 27 | 29 29 |
| 0x00002b0 | 31 31 | 03 03 | 05 05 | 45 45 | 47 47 | 13 13 | 15 15 | 37 37 |
| 0x00002d0 | 39 39 | 05 05 | 07 07 | 53 53 | 55 55 | 21 21 | 23 23 | 33 33 |
| 0x00002f0 | 35 35 | 01 01 | 03 03 | 49 49 | 75 75 | 39 39 | 43 43 | 07 07 |
| 0x0000310 | 11 11 | 47 47 | 51 51 | 15 15 | 19 19 | 55 55 | 59 59 | 61 61 |
| 0x0000330 | 65 65 | 29 29 | 33 33 | 69 69 | 73 73 | 37 37 | 41 41 | 77 77 |
| 0x0000350 | 09 09 | 45 45 | 49 49 | 13 13 | 17 17 | 53 53 | 57 57 | 63 63 |
| 0x0000370 | 67 67 | 31 31 | 35 35 | 71 71 | 35 35 | 73 73 | 75 75 | 41 41 |
| 0x0000390 | 43 43 | 09 09 | 11 11 | 49 49 | 51 51 | 17 17 | 19 19 | 21 21 |
| 0x00003b0 | 23 23 | 61 61 | 63 63 | 29 29 | 31 31 | 69 69 | 71 71 | 37 37 |
| 0x00003d0 | 39 39 | 77 77 | 07 07 | 45 45 | 47 47 | 13 13 | 15 15 | 25 25 |
| 0x00003f0 | 27 27 | 65 65 | 67 67 | 33 33 | 51 51 | 39 39 | 43 43 | 77 77 |

## 2.3 THIRD SET

```
Hop sequence {k} for PAGE SCAN/INQUIRY SCAN SUBSTATE:
CLKN start:     0x0000000
ULAP:           0x6587cba9
#ticks:         0000 | 1000 | 2000 | 3000 | 4000 | 5000 | 6000 | 7000 |
                -----------------------------------------------------
0x0000000:        16 |   65 |   67 |   18 |   20 |   53 |   55 |    6 |
0x0008000:         8 |   57 |   59 |   10 |   12 |   69 |   71 |   22 |
0x0010000:        24 |   73 |   75 |   26 |   28 |   45 |   47 |   77 |
0x0018000:         0 |   49 |   51 |    2 |    4 |   61 |   63 |   14 |
0x0020000:        16 |   65 |   67 |   18 |   20 |   53 |   55 |    6 |
0x0028000:         8 |   57 |   59 |   10 |   12 |   69 |   71 |   22 |
0x0030000:        24 |   73 |   75 |   26 |   28 |   45 |   47 |   77 |
0x0038000:         0 |   49 |   51 |    2 |    4 |   61 |   63 |   14 |

Hop sequence {k} for PAGE STATE/INQUIRY SUBSTATE:
CLKE start:     0x0000000
ULAP:           0x6587cba9
#ticks:         00 01 02 03 | 04 05 06 07 | 08 09 0a 0b | 0c 0d 0e 0f |
                -----------------------------------------------------
0x0000000:      00 49 36 38 | 51 02 42 40 | 04 61 44 46 | 63 14 50 48 |
0x0000010:      16 65 52 54 | 67 18 58 56 | 20 53 60 62 | 55 06 66 64 |
0x0000020:      00 49 36 38 | 51 02 42 40 | 04 61 44 46 | 63 14 50 48 |
0x0000030:      16 65 52 54 | 67 18 58 56 | 20 53 60 62 | 55 06 66 64 |
...
0x0001000:      00 57 36 70 | 59 10 74 72 | 12 69 76 78 | 71 22 03 01 |
0x0001010:      24 73 05 07 | 75 26 11 09 | 28 45 13 30 | 47 77 34 32 |
0x0001020:      00 57 36 70 | 59 10 74 72 | 12 69 76 78 | 71 22 03 01 |
0x0001030:      24 73 05 07 | 75 26 11 09 | 28 45 13 30 | 47 77 34 32 |
...
0x0002000:      08 57 68 70 | 51 02 42 40 | 04 61 44 46 | 63 14 50 48 |
0x0002010:      16 65 52 54 | 67 18 58 56 | 20 53 60 62 | 55 06 66 64 |
0x0002020:      08 57 68 70 | 51 02 42 40 | 04 61 44 46 | 63 14 50 48 |
0x0002030:      16 65 52 54 | 67 18 58 56 | 20 53 60 62 | 55 06 66 64 |
...
0x0003000:      00 49 36 38 | 51 10 42 72 | 12 69 76 78 | 71 22 03 01 |
0x0003010:      24 73 05 07 | 75 26 11 09 | 28 45 13 30 | 47 77 34 32 |
0x0003020:      00 49 36 38 | 51 10 42 72 | 12 69 76 78 | 71 22 03 01 |
0x0003030:      24 73 05 07 | 75 26 11 09 | 28 45 13 30 | 47 77 34 32 |

Hop sequence {k} for SLAVE PAGE RESPONSE SUBSTATE:
CLKN* =         0x0000010
ULAP:           0x6587cba9
#ticks:         00 | 02 04 | 06 08 | 0a 0c | 0e 10 | 12 14 | 16 18 | 1a 1c | 1e
                -----------------------------------------------------------------
0x0000012:      52 | 65 54 | 67 58 | 18 56 | 20 60 | 53 62 | 55 66 | 06 64 | 08
0x0000032:      68 | 57 70 | 59 74 | 10 72 | 12 76 | 69 78 | 71 03 | 22 01 | 24
0x0000052:      05 | 73 07 | 75 11 | 26 09 | 28 13 | 45 30 | 47 34 | 77 32 | 00
0x0000072:      36 | 49 38 | 51 42 | 02 40 | 04 44 | 61 46 | 63 50 | 14 48 | 16

Hop sequence {k} for MASTER PAGE RESPONSE SUBSTATE:
Offset value:   24
CLKE* =         0x0000012
ULAP:           0x6587cba9
#ticks:         00 02 | 04 06 | 08 0a | 0c 0e | 10 12 | 14 16 | 18 1a | 1c 1e |
                -----------------------------------------------------------------
0x0000014:      65 54 | 67 58 | 18 56 | 20 60 | 53 62 | 55 66 | 06 64 | 08 68 |
0x0000034:      57 70 | 59 74 | 10 72 | 12 76 | 69 78 | 71 03 | 22 01 | 24 05 |
0x0000054:      73 07 | 75 11 | 26 09 | 28 13 | 45 30 | 47 34 | 77 32 | 00 36 |
0x0000074:      49 38 | 51 42 | 02 40 | 04 44 | 61 46 | 63 50 | 14 48 | 16 52 |
                                    Page 1
```

Hop sequence {k} for CONNECTION STATE (Basic channel hopping sequence; ie, non-AFH):
CLK start: 0x0000010
ULAP: 0x6587cba9

| #ticks: | 00 02 | 04 06 | 08 0a | 0c 0e | 10 12 | 14 16 | 18 1a | 1c 1e |
|---|---|---|---|---|---|---|---|---|
| 0x0000010: | 20 60 | 53 62 | 55 66 | 06 64 | 08 68 | 57 70 | 59 74 | 10 72 |
| 0x0000030: | 12 76 | 69 78 | 71 03 | 22 01 | 24 05 | 73 07 | 75 11 | 26 09 |
| 0x0000050: | 28 13 | 45 30 | 47 34 | 77 32 | 00 36 | 49 38 | 51 42 | 02 40 |
| 0x0000070: | 04 44 | 61 46 | 63 50 | 14 48 | 50 05 | 16 07 | 20 09 | 48 11 |
| 0x0000090: | 52 13 | 06 15 | 10 17 | 38 19 | 42 21 | 08 23 | 12 25 | 40 27 |
| 0x00000b0: | 44 29 | 22 31 | 26 33 | 54 35 | 58 37 | 24 39 | 28 41 | 56 43 |
| 0x00000d0: | 60 45 | 77 62 | 02 64 | 30 66 | 34 68 | 00 70 | 04 72 | 32 74 |
| 0x00000f0: | 36 76 | 14 78 | 18 01 | 46 03 | 72 29 | 42 39 | 44 43 | 74 41 |
| 0x0000110: | 76 45 | 46 47 | 48 51 | 78 49 | 01 53 | 50 63 | 52 67 | 03 65 |
| 0x0000130: | 05 69 | 54 55 | 56 59 | 07 57 | 09 61 | 58 71 | 60 75 | 11 73 |
| 0x0000150: | 13 77 | 30 15 | 32 19 | 62 17 | 64 21 | 34 31 | 36 35 | 66 33 |
| 0x0000170: | 68 37 | 38 23 | 40 27 | 70 25 | 27 61 | 72 71 | 76 73 | 25 75 |
| 0x0000190: | 29 77 | 78 00 | 03 02 | 31 04 | 35 06 | 01 16 | 05 18 | 33 20 |
| 0x00001b0: | 37 22 | 07 08 | 11 10 | 39 12 | 43 14 | 09 24 | 13 26 | 41 28 |
| 0x00001d0: | 45 30 | 62 47 | 66 49 | 15 51 | 19 53 | 64 63 | 68 65 | 17 67 |
| 0x00001f0: | 21 69 | 70 55 | 74 57 | 23 59 | 53 22 | 35 12 | 37 28 | 67 14 |
| 0x0000210: | 69 30 | 23 32 | 25 48 | 55 34 | 57 50 | 39 40 | 41 56 | 71 42 |
| 0x0000230: | 73 58 | 27 36 | 29 52 | 59 38 | 61 54 | 43 44 | 45 60 | 75 46 |
| 0x0000250: | 77 62 | 15 00 | 17 16 | 47 02 | 49 18 | 31 08 | 33 24 | 63 10 |
| 0x0000270: | 65 26 | 19 04 | 21 20 | 51 06 | 06 54 | 65 42 | 69 58 | 18 46 |
| 0x0000290: | 22 62 | 55 64 | 59 01 | 08 68 | 12 05 | 71 72 | 75 09 | 24 76 |
| 0x00002b0: | 28 13 | 57 66 | 61 03 | 10 70 | 14 07 | 73 74 | 77 11 | 26 78 |
| 0x00002d0: | 30 15 | 47 32 | 51 48 | 00 36 | 04 52 | 63 40 | 67 56 | 16 44 |
| 0x00002f0: | 20 60 | 49 34 | 53 50 | 02 38 | 38 78 | 12 05 | 14 13 | 44 07 |
| 0x0000310: | 46 15 | 16 17 | 18 25 | 48 19 | 50 27 | 24 33 | 26 41 | 56 35 |
| 0x0000330: | 58 43 | 20 21 | 22 29 | 52 23 | 54 31 | 28 37 | 30 45 | 60 39 |
| 0x0000350: | 62 47 | 00 64 | 02 72 | 32 66 | 34 74 | 08 01 | 10 09 | 40 03 |
| 0x0000370: | 42 11 | 04 68 | 06 76 | 36 70 | 70 31 | 42 35 | 46 43 | 74 39 |
| 0x0000390: | 78 47 | 48 49 | 52 57 | 01 53 | 05 61 | 56 65 | 60 73 | 09 69 |
| 0x00003b0: | 13 77 | 50 51 | 54 59 | 03 55 | 07 63 | 58 67 | 62 75 | 11 71 |
| 0x00003d0: | 15 00 | 32 17 | 36 25 | 64 21 | 68 29 | 40 33 | 44 41 | 72 37 |
| 0x00003f0: | 76 45 | 34 19 | 38 27 | 66 23 | 11 71 | 05 18 | 07 22 | 13 20 |

*Sample Data*

Hop Sequence {k} for CONNECTION STATE (Adapted channel hopping sequence with all channel used; ie, AFH(79)):

CLK start: 0x0000010

ULAP: 0x6587cba9

Used Channels:0x7fffffffffffffffff

| #ticks: | 00 02 | 04 06 | 08 0a | 0c 0e | 10 12 | 14 16 | 18 1a | 1c 1e |
|---|---|---|---|---|---|---|---|---|
| 0x0000010 | 20 20 | 53 53 | 55 55 | 06 06 | 08 08 | 57 57 | 59 59 | 10 10 |
| 0x0000030 | 12 12 | 69 69 | 71 71 | 22 22 | 24 24 | 73 73 | 75 75 | 26 26 |
| 0x0000050 | 28 28 | 45 45 | 47 47 | 77 77 | 00 00 | 49 49 | 51 51 | 02 02 |
| 0x0000070 | 04 04 | 61 61 | 63 63 | 14 14 | 50 50 | 16 16 | 20 20 | 48 48 |
| 0x0000090 | 52 52 | 06 06 | 10 10 | 38 38 | 42 42 | 08 08 | 12 12 | 40 40 |
| 0x00000b0 | 44 44 | 22 22 | 26 26 | 54 54 | 58 58 | 24 24 | 28 28 | 56 56 |
| 0x00000d0 | 60 60 | 77 77 | 02 02 | 30 30 | 34 34 | 00 00 | 04 04 | 32 32 |
| 0x00000f0 | 36 36 | 14 14 | 18 18 | 46 46 | 72 72 | 42 42 | 44 44 | 74 74 |
| 0x0000110 | 76 76 | 46 46 | 48 48 | 78 78 | 01 01 | 50 50 | 52 52 | 03 03 |
| 0x0000130 | 05 05 | 54 54 | 56 56 | 07 07 | 09 09 | 58 58 | 60 60 | 11 11 |
| 0x0000150 | 13 13 | 30 30 | 32 32 | 62 62 | 64 64 | 34 34 | 36 36 | 66 66 |
| 0x0000170 | 68 68 | 38 38 | 40 40 | 70 70 | 27 27 | 72 72 | 76 76 | 25 25 |
| 0x0000190 | 29 29 | 78 78 | 03 03 | 31 31 | 35 35 | 01 01 | 05 05 | 33 33 |
| 0x00001b0 | 37 37 | 07 07 | 11 11 | 39 39 | 43 43 | 09 09 | 13 13 | 41 41 |
| 0x00001d0 | 45 45 | 62 62 | 66 66 | 15 15 | 19 19 | 64 64 | 68 68 | 17 17 |
| 0x00001f0 | 21 21 | 70 70 | 74 74 | 23 23 | 53 53 | 35 35 | 37 37 | 67 67 |
| 0x0000210 | 69 69 | 23 23 | 25 25 | 55 55 | 57 57 | 39 39 | 41 41 | 71 71 |
| 0x0000230 | 73 73 | 27 27 | 29 29 | 59 59 | 61 61 | 43 43 | 45 45 | 75 75 |
| 0x0000250 | 77 77 | 15 15 | 17 17 | 47 47 | 49 49 | 31 31 | 33 33 | 63 63 |
| 0x0000270 | 65 65 | 19 19 | 21 21 | 51 51 | 06 06 | 65 65 | 69 69 | 18 18 |
| 0x0000290 | 22 22 | 55 55 | 59 59 | 08 08 | 12 12 | 71 71 | 75 75 | 24 24 |
| 0x00002b0 | 28 28 | 57 57 | 61 61 | 10 10 | 14 14 | 73 73 | 77 77 | 26 26 |
| 0x00002d0 | 30 30 | 47 47 | 51 51 | 00 00 | 04 04 | 63 63 | 67 67 | 16 16 |
| 0x00002f0 | 20 20 | 49 49 | 53 53 | 02 02 | 38 38 | 12 12 | 14 14 | 44 44 |
| 0x0000310 | 46 46 | 16 16 | 18 18 | 48 48 | 50 50 | 24 24 | 26 26 | 56 56 |
| 0x0000330 | 58 58 | 20 20 | 22 22 | 52 52 | 54 54 | 28 28 | 30 30 | 60 60 |
| 0x0000350 | 62 62 | 00 00 | 02 02 | 32 32 | 34 34 | 08 08 | 10 10 | 40 40 |
| 0x0000370 | 42 42 | 04 04 | 06 06 | 36 36 | 70 70 | 42 42 | 46 46 | 74 74 |
| 0x0000390 | 78 78 | 48 48 | 52 52 | 01 01 | 05 05 | 56 56 | 60 60 | 09 09 |
| 0x00003b0 | 13 13 | 50 50 | 54 54 | 03 03 | 07 07 | 58 58 | 62 62 | 11 11 |
| 0x00003d0 | 15 15 | 32 32 | 36 36 | 64 64 | 68 68 | 40 40 | 44 44 | 72 72 |
| 0x00003f0 | 76 76 | 34 34 | 38 38 | 66 66 | 11 11 | 05 05 | 07 07 | 13 13 |

*Sample Data*

Hop Sequence {k} for CONNECTION STATE (Adapted channel hopping sequence with channels 0 to 21 unused):

CLK start: 0x0000010

ULAP: 0x6587cba9

Used Channels:0x7fffffffffffc00000

| #ticks: | 00 02 | 04 06 | 08 0a | 0c 0e | 10 12 | 14 16 | 18 1a | 1c 1e |
|---|---|---|---|---|---|---|---|---|
| 0x0000010 | 29 29 | 53 53 | 55 55 | 72 72 | 74 74 | 57 57 | 59 59 | 76 76 |
| 0x0000030 | 78 78 | 69 69 | 71 71 | 22 22 | 24 24 | 73 73 | 75 75 | 26 26 |
| 0x0000050 | 28 28 | 45 45 | 47 47 | 77 77 | 66 66 | 49 49 | 51 51 | 68 68 |
| 0x0000070 | 70 70 | 61 61 | 63 63 | 23 23 | 50 50 | 25 25 | 29 29 | 48 48 |
| 0x0000090 | 52 52 | 72 72 | 76 76 | 38 38 | 42 42 | 74 74 | 78 78 | 40 40 |
| 0x00000b0 | 44 44 | 22 22 | 26 26 | 54 54 | 58 58 | 24 24 | 28 28 | 56 56 |
| 0x00000d0 | 60 60 | 77 77 | 68 68 | 30 30 | 34 34 | 66 66 | 70 70 | 32 32 |
| 0x00000f0 | 36 36 | 23 23 | 27 27 | 46 46 | 72 72 | 42 42 | 44 44 | 74 74 |
| 0x0000110 | 76 76 | 46 46 | 48 48 | 78 78 | 32 32 | 50 50 | 52 52 | 34 34 |
| 0x0000130 | 36 36 | 54 54 | 56 56 | 38 38 | 40 40 | 58 58 | 60 60 | 42 42 |
| 0x0000150 | 44 44 | 30 30 | 32 32 | 62 62 | 64 64 | 34 34 | 36 36 | 66 66 |
| 0x0000170 | 68 68 | 38 38 | 40 40 | 70 70 | 27 27 | 72 72 | 76 76 | 25 25 |
| 0x0000190 | 29 29 | 78 78 | 34 34 | 31 31 | 35 35 | 32 32 | 36 36 | 33 33 |
| 0x00001b0 | 37 37 | 38 38 | 42 42 | 39 39 | 43 43 | 40 40 | 44 44 | 41 41 |
| 0x00001d0 | 45 45 | 62 62 | 66 66 | 46 46 | 50 50 | 64 64 | 68 68 | 48 48 |
| 0x00001f0 | 52 52 | 70 70 | 74 74 | 23 23 | 53 53 | 35 35 | 37 37 | 67 67 |
| 0x0000210 | 69 69 | 23 23 | 25 25 | 55 55 | 57 57 | 39 39 | 41 41 | 71 71 |
| 0x0000230 | 73 73 | 27 27 | 29 29 | 59 59 | 61 61 | 43 43 | 45 45 | 75 75 |
| 0x0000250 | 77 77 | 46 46 | 48 48 | 47 47 | 49 49 | 31 31 | 33 33 | 63 63 |
| 0x0000270 | 65 65 | 50 50 | 52 52 | 51 51 | 59 59 | 65 65 | 69 69 | 71 71 |
| 0x0000290 | 22 22 | 55 55 | 59 59 | 61 61 | 65 65 | 71 71 | 75 75 | 24 24 |
| 0x00002b0 | 28 28 | 57 57 | 61 61 | 63 63 | 67 67 | 73 73 | 77 77 | 26 26 |
| 0x00002d0 | 30 30 | 47 47 | 51 51 | 53 53 | 57 57 | 63 63 | 67 67 | 69 69 |
| 0x00002f0 | 73 73 | 49 49 | 53 53 | 55 55 | 38 38 | 65 65 | 67 67 | 44 44 |
| 0x0000310 | 46 46 | 69 69 | 71 71 | 48 48 | 50 50 | 24 24 | 26 26 | 56 56 |
| 0x0000330 | 58 58 | 73 73 | 22 22 | 52 52 | 54 54 | 28 28 | 30 30 | 60 60 |
| 0x0000350 | 62 62 | 53 53 | 55 55 | 32 32 | 34 34 | 61 61 | 63 63 | 40 40 |
| 0x0000370 | 42 42 | 57 57 | 59 59 | 36 36 | 70 70 | 42 42 | 46 46 | 74 74 |
| 0x0000390 | 78 78 | 48 48 | 52 52 | 76 76 | 23 23 | 56 56 | 60 60 | 27 27 |
| 0x00003b0 | 31 31 | 50 50 | 54 54 | 78 78 | 25 25 | 58 58 | 62 62 | 29 29 |
| 0x00003d0 | 33 33 | 32 32 | 36 36 | 64 64 | 68 68 | 40 40 | 44 44 | 72 72 |
| 0x00003f0 | 76 76 | 34 34 | 38 38 | 66 66 | 29 29 | 23 23 | 25 25 | 31 31 |

*Sample Data*

Hop Sequence {k} for CONNECTION STATE (Adapted channel hopping sequence with even channels used):

CLK start: 0x0000010

ULAP: 0x6587cba9

Used Channels:0x55555555555555555

| #ticks: | 00 02 | 04 06 | 08 0a | 0c 0e | 10 12 | 14 16 | 18 1a | 1c 1e |
|---|---|---|---|---|---|---|---|---|
| 0x0000010 | 20 20 | 52 52 | 54 54 | 06 06 | 08 08 | 56 56 | 58 58 | 10 10 |
| 0x0000030 | 12 12 | 68 68 | 70 70 | 22 22 | 24 24 | 72 72 | 74 74 | 26 26 |
| 0x0000050 | 28 28 | 44 44 | 46 46 | 76 76 | 00 00 | 48 48 | 50 50 | 02 02 |
| 0x0000070 | 04 04 | 60 60 | 62 62 | 14 14 | 50 50 | 16 16 | 20 20 | 48 48 |
| 0x0000090 | 52 52 | 06 06 | 10 10 | 38 38 | 42 42 | 08 08 | 12 12 | 40 40 |
| 0x00000b0 | 44 44 | 22 22 | 26 26 | 54 54 | 58 58 | 24 24 | 28 28 | 56 56 |
| 0x00000d0 | 60 60 | 76 76 | 02 02 | 30 30 | 34 34 | 00 00 | 04 04 | 32 32 |
| 0x00000f0 | 36 36 | 14 14 | 18 18 | 46 46 | 72 72 | 42 42 | 44 44 | 74 74 |
| 0x0000110 | 76 76 | 46 46 | 48 48 | 78 78 | 78 78 | 50 50 | 52 52 | 00 00 |
| 0x0000130 | 02 02 | 54 54 | 56 56 | 04 04 | 06 06 | 58 58 | 60 60 | 08 08 |
| 0x0000150 | 10 10 | 30 30 | 32 32 | 62 62 | 64 64 | 34 34 | 36 36 | 66 66 |
| 0x0000170 | 68 68 | 38 38 | 40 40 | 70 70 | 24 24 | 72 72 | 76 76 | 22 22 |
| 0x0000190 | 26 26 | 78 78 | 00 00 | 28 28 | 32 32 | 78 78 | 02 02 | 30 30 |
| 0x00001b0 | 34 34 | 04 04 | 08 08 | 36 36 | 40 40 | 06 06 | 10 10 | 38 38 |
| 0x00001d0 | 42 42 | 62 62 | 66 66 | 12 12 | 16 16 | 64 64 | 68 68 | 14 14 |
| 0x00001f0 | 18 18 | 70 70 | 74 74 | 20 20 | 50 50 | 32 32 | 34 34 | 64 64 |
| 0x0000210 | 66 66 | 20 20 | 22 22 | 52 52 | 54 54 | 36 36 | 38 38 | 68 68 |
| 0x0000230 | 70 70 | 24 24 | 26 26 | 56 56 | 58 58 | 40 40 | 42 42 | 72 72 |
| 0x0000250 | 74 74 | 12 12 | 14 14 | 44 44 | 46 46 | 28 28 | 30 30 | 60 60 |
| 0x0000270 | 62 62 | 16 16 | 18 18 | 48 48 | 06 06 | 62 62 | 66 66 | 18 18 |
| 0x0000290 | 22 22 | 52 52 | 56 56 | 08 08 | 12 12 | 68 68 | 72 72 | 24 24 |
| 0x00002b0 | 28 28 | 54 54 | 58 58 | 10 10 | 14 14 | 70 70 | 74 74 | 26 26 |
| 0x00002d0 | 30 30 | 44 44 | 48 48 | 00 00 | 04 04 | 60 60 | 64 64 | 16 16 |
| 0x00002f0 | 20 20 | 46 46 | 50 50 | 02 02 | 38 38 | 12 12 | 14 14 | 44 44 |
| 0x0000310 | 46 46 | 16 16 | 18 18 | 48 48 | 50 50 | 24 24 | 26 26 | 56 56 |
| 0x0000330 | 58 58 | 20 20 | 22 22 | 52 52 | 54 54 | 28 28 | 30 30 | 60 60 |
| 0x0000350 | 62 62 | 00 00 | 02 02 | 32 32 | 34 34 | 08 08 | 10 10 | 40 40 |
| 0x0000370 | 42 42 | 04 04 | 06 06 | 36 36 | 70 70 | 42 42 | 46 46 | 74 74 |
| 0x0000390 | 78 78 | 48 48 | 52 52 | 76 76 | 00 00 | 56 56 | 60 60 | 04 04 |
| 0x00003b0 | 08 08 | 50 50 | 54 54 | 78 78 | 02 02 | 58 58 | 62 62 | 06 06 |
| 0x00003d0 | 10 10 | 32 32 | 36 36 | 64 64 | 68 68 | 40 40 | 44 44 | 72 72 |
| 0x00003f0 | 76 76 | 34 34 | 38 38 | 66 66 | 06 06 | 00 00 | 02 02 | 08 08 |

*Sample Data*

Hop Sequence {k} for CONNECTION STATE (Adapted channel hopping sequence with odd channels used):

CLK start: 0x0000010

ULAP: 0x6587cba9

Used Channels:0x2aaaaaaaaaaaaaaaaaaa

| #ticks: | 00 02 | 04 06 | 08 0a | 0c 0e | 10 12 | 14 16 | 18 1a | 1c 1e |
|---|---|---|---|---|---|---|---|---|
| 0x0000010 | 23 23 | 53 53 | 55 55 | 09 09 | 11 11 | 57 57 | 59 59 | 13 13 |
| 0x0000030 | 15 15 | 69 69 | 71 71 | 25 25 | 27 27 | 73 73 | 75 75 | 29 29 |
| 0x0000050 | 31 31 | 45 45 | 47 47 | 77 77 | 03 03 | 49 49 | 51 51 | 05 05 |
| 0x0000070 | 07 07 | 61 61 | 63 63 | 17 17 | 53 53 | 19 19 | 23 23 | 51 51 |
| 0x0000090 | 55 55 | 09 09 | 13 13 | 41 41 | 45 45 | 11 11 | 15 15 | 43 43 |
| 0x00000b0 | 47 47 | 25 25 | 29 29 | 57 57 | 61 61 | 27 27 | 31 31 | 59 59 |
| 0x00000d0 | 63 63 | 77 77 | 05 05 | 33 33 | 37 37 | 03 03 | 07 07 | 35 35 |
| 0x00000f0 | 39 39 | 17 17 | 21 21 | 49 49 | 75 75 | 45 45 | 47 47 | 77 77 |
| 0x0000110 | 01 01 | 49 49 | 51 51 | 03 03 | 01 01 | 53 53 | 55 55 | 03 03 |
| 0x0000130 | 05 05 | 57 57 | 59 59 | 07 07 | 09 09 | 61 61 | 63 63 | 11 11 |
| 0x0000150 | 13 13 | 33 33 | 35 35 | 65 65 | 67 67 | 37 37 | 39 39 | 69 69 |
| 0x0000170 | 71 71 | 41 41 | 43 43 | 73 73 | 27 27 | 75 75 | 01 01 | 25 25 |
| 0x0000190 | 29 29 | 03 03 | 03 03 | 31 31 | 35 35 | 01 01 | 05 05 | 33 33 |
| 0x00001b0 | 37 37 | 07 07 | 11 11 | 39 39 | 43 43 | 09 09 | 13 13 | 41 41 |
| 0x00001d0 | 45 45 | 65 65 | 69 69 | 15 15 | 19 19 | 67 67 | 71 71 | 17 17 |
| 0x00001f0 | 21 21 | 73 73 | 77 77 | 23 23 | 53 53 | 35 35 | 37 37 | 67 67 |
| 0x0000210 | 69 69 | 23 23 | 25 25 | 55 55 | 57 57 | 39 39 | 41 41 | 71 71 |
| 0x0000230 | 73 73 | 27 27 | 29 29 | 59 59 | 61 61 | 43 43 | 45 45 | 75 75 |
| 0x0000250 | 77 77 | 15 15 | 17 17 | 47 47 | 49 49 | 31 31 | 33 33 | 63 63 |
| 0x0000270 | 65 65 | 19 19 | 21 21 | 51 51 | 11 11 | 65 65 | 69 69 | 23 23 |
| 0x0000290 | 27 27 | 55 55 | 59 59 | 13 13 | 17 17 | 71 71 | 75 75 | 29 29 |
| 0x00002b0 | 33 33 | 57 57 | 61 61 | 15 15 | 19 19 | 73 73 | 77 77 | 31 31 |
| 0x00002d0 | 35 35 | 47 47 | 51 51 | 05 05 | 09 09 | 63 63 | 67 67 | 21 21 |
| 0x00002f0 | 25 25 | 49 49 | 53 53 | 07 07 | 43 43 | 17 17 | 19 19 | 49 49 |
| 0x0000310 | 51 51 | 21 21 | 23 23 | 53 53 | 55 55 | 29 29 | 31 31 | 61 61 |
| 0x0000330 | 63 63 | 25 25 | 27 27 | 57 57 | 59 59 | 33 33 | 35 35 | 65 65 |
| 0x0000350 | 67 67 | 05 05 | 07 07 | 37 37 | 39 39 | 13 13 | 15 15 | 45 45 |
| 0x0000370 | 47 47 | 09 09 | 11 11 | 41 41 | 75 75 | 47 47 | 51 51 | 01 01 |
| 0x0000390 | 05 05 | 53 53 | 57 57 | 01 01 | 05 05 | 61 61 | 65 65 | 09 09 |
| 0x00003b0 | 13 13 | 55 55 | 59 59 | 03 03 | 07 07 | 63 63 | 67 67 | 11 11 |
| 0x00003d0 | 15 15 | 37 37 | 41 41 | 69 69 | 73 73 | 45 45 | 49 49 | 77 77 |
| 0x00003f0 | 03 03 | 39 39 | 43 43 | 71 71 | 11 11 | 05 05 | 07 07 | 13 13 |

# 3 ACCESS CODE SAMPLE DATA

## Different access codes (GIAC, DIACs, others...)

LAP with LSB as rightmost bit.

```
                  Bit transmit order on air
                ------------------------------->
LAP:        | Preamble:| Sync word:      | Trailer: |
-----------------------------------------------------
000000      |   5    | 7e7041e3 4000000d |    5     |
ffffff      |   a    | e758b522 7ffffff2 |    a     |
9e8b33      |   5    | 475c58cc 73345e72 |    a     |
9e8b34      |   5    | 28ed3c34 cb345e72 |    a     |
9e8b36      |   5    | 62337b64 1b345e72 |    a     |
9e8b39      |   a    | c05747b9 e7345e72 |    a     |
9e8b3d      |   5    | 7084eab0 2f345e72 |    a     |
9e8b42      |   5    | 64c86d2b 90b45e72 |    a     |
9e8b48      |   a    | e3c3725e 04b45e72 |    a     |
9e8b4f      |   a    | 8c7216a6 bcb45e72 |    a     |
9e8b57      |   a    | b2f16c30 fab45e72 |    a     |
9e8b60      |   5    | 57bd3b22 c1b45e72 |    a     |
9e8b6a      |   a    | d0b62457 55b45e72 |    a     |
9e8b75      |   a    | 81843a39 abb45e72 |    a     |
9e8b81      |   5    | 0ca96681 e0745e72 |    a     |
9e8b8e      |   a    | aecd5a5c 1c745e72 |    a     |
9e8b9c      |   5    | 17453fbf ce745e72 |    a     |
9e8bab      |   a    | f20968ad f5745e72 |    a     |
9e8bbb      |   5    | 015f4a1e f7745e72 |    a     |
9e8bcc      |   a    | d8c695a0 0cf45e72 |    a     |
9e8bde      |   5    | 614ef043 def45e72 |    a     |
9e8bf1      |   a    | ba81ddc7 a3f45e72 |    a     |
9e8c05      |   5    | 64a7dc4f 680c5e72 |    a     |
9e8c1a      |   5    | 3595c221 960c5e72 |    a     |
9e8c30      |   a    | cb35cc0d 830c5e72 |    a     |
9e8c47      |   5    | 12ac13b3 788c5e72 |    a     |
9e8c5f      |   5    | 2c2f6925 3e8c5e72 |    a     |
9e8c78      |   5    | 3a351c84 078c5e72 |    a     |
9e8c92      |   5    | 7396d0f3 124c5e72 |    a     |
9e8cad      |   5    | 5b0fdfc4 6d4c5e72 |    a     |
9e8cc9      |   a    | aea2eb38 e4cc5e72 |    a     |
9e8ce6      |   5    | 756dc6bc 99cc5e72 |    a     |
9e8d04      |   5    | 214cf934 882c5e72 |    a     |
9e8d23      |   5    | 37568c95 b12c5e72 |    a     |
9e8d43      |   5    | 72281560 f0ac5e72 |    a     |
9e8d64      |   5    | 643260c1 c9ac5e72 |    a     |
9e8d86      |   a    | e044f493 986c5e72 |    a     |
9e8da9      |   5    | 3b8bd917 e56c5e72 |    a     |
9e8dcd      |   a    | ce26edeb 6cec5e72 |    a     |
9e8df2      |   a    | e6bfe2dc 13ec5e72 |    a     |
9e8e18      |   a    | 82dcde3d c61c5e72 |    a     |
9e8e3f      |   a    | 94c6ab9c ff1c5e72 |    a     |
```

*Sample Data*

| | | | |
|---|---|---|---|
| 9e8e67 | a | 969059a6 799c5e72 | a |
| 9e8e90 | a | c4dfccef 425c5e72 | a |
| 9e8eba | 5 | 3a7fc2c3 575c5e72 | a |
| 9e8ee5 | 5 | 57985401 69dc5e72 | a |
| 9e8f11 | 5 | 0ae2a363 623c5e72 | a |
| 9e8f3e | a | d12d8ee7 1f3c5e72 | a |
| 9e8f6c | 5 | 547063a8 0dbc5e72 | a |
| 9e8f9b | 5 | 063ff6e1 367c5e72 | a |
| 9e8fcb | a | c9bc5cfe f4fc5e72 | a |
| 9e8ffc | 5 | 2cf00bec cffc5e72 | a |
| 9e902e | a | 8ec5052f 5d025e72 | a |
| 9e9061 | 5 | 1074b15e 61825e72 | a |
| 9e9095 | a | 9d59ede6 2a425e72 | a |
| 9e90ca | a | f0be7b24 14c25e72 | a |
| 9e9100 | 5 | 10e10dd0 c0225e72 | a |
| 9e9137 | a | f5ad5ac2 fb225e72 | a |
| 9e916f | a | f7fba8f8 7da25e72 | a |
| 9e91a8 | 5 | 2f490e5b c5625e72 | a |
| 9e91e2 | a | 94979982 91e25e72 | a |
| 9e921d | 5 | 26cda478 2e125e72 | a |
| 9e9259 | a | aacb81dd 26925e72 | a |
| 9e9296 | a | bfac7f5b da525e72 | a |
| 9e92d4 | a | c9a7b0a7 cad25e72 | a |
| 9e9313 | a | c142bdde 32325e72 | a |
| 616cec | 5 | 586a491f 0dcda18d | 5 |
| 616ceb | 5 | 37db2de7 b5cda18d | 5 |
| 616ce9 | 5 | 7d056ab7 65cda18d | 5 |
| 616ce6 | a | df61566a 99cda18d | 5 |
| 616ce2 | 5 | 6fb2fb63 51cda18d | 5 |
| 616cdd | 5 | 472bf454 2ecda18d | 5 |
| 616cd7 | a | c020eb21 bacda18d | 5 |
| 616cd0 | a | af918fd9 02cda18d | 5 |
| 616cc8 | a | 9112f54f 44cda18d | 5 |
| 616cbf | 5 | 488b2af1 bf4da18d | 5 |
| 616cb5 | a | cf803584 2b4da18d | 5 |
| 616caa | a | 9eb22bea d54da18d | 5 |
| 616c9e | a | a49cb509 9e4da18d | 5 |
| 616c91 | 5 | 06f889d4 624da18d | 5 |
| 616c83 | a | bf70ec37 b04da18d | 5 |
| 616c74 | a | ed3f797e 8b8da18d | 5 |
| 616c64 | 5 | 1e695bcd 898da18d | 5 |
| 616c53 | a | fb250cdf b28da18d | 5 |
| 616c41 | 5 | 42ad693c 608da18d | 5 |
| 616c2e | a | a5b7cc14 dd0da18d | 5 |
| 616c1a | a | 9f9952f7 960da18d | 5 |
| 616c05 | a | ceab4c99 680da18d | 5 |
| 616bef | a | d403ddde fdf5a18d | 5 |
| 616bd8 | 5 | 314f8acc c6f5a18d | 5 |
| 616bc0 | 5 | 0fccf05a 80f5a18d | 5 |
| 616ba7 | 5 | 25030d57 7975a18d | 5 |
| 616b8d | a | dba3037b 6c75a18d | 5 |
| 616b72 | 5 | 4439ce17 13b5a18d | 5 |

*Sample Data*

| | | | |
|---|---|---|---|
| 616b56 | a | 8d417247 5ab5a18d | 5 |
| 616b39 | 5 | 6a5bd76f e735a18d | 5 |
| 616b1b | 5 | 592e8166 b635a18d | 5 |
| 616afc | 5 | 28609d46 cfd5a18d | 5 |
| 616adc | 5 | 51cb8c1f 4ed5a18d | 5 |
| 616abb | 5 | 7b047112 b755a18d | 5 |
| 616a99 | 5 | 4871271b e655a18d | 5 |
| 616a76 | 5 | 24bdc8c4 9b95a18d | 5 |
| 616a52 | a | edc57494 d295a18d | 5 |
| 616a2d | a | f989f30f 6d15a18d | 5 |
| 616a07 | 5 | 0729fd23 7815a18d | 5 |
| 6169e0 | a | 8bf0ba4f 81e5a18d | 5 |
| 6169b8 | a | 89a64875 0765a18d | 5 |
| 61698f | 5 | 6cea1f67 3c65a18d | 5 |
| 616965 | 5 | 2549d310 29a5a18d | 5 |
| 61693a | 5 | 48ae45d2 1725a18d | 5 |
| 61690e | 5 | 7280db31 5c25a18d | 5 |
| 6168e1 | a | ce1b9f34 61c5a18d | 5 |
| 6168b3 | 5 | 4b46727b 7345a18d | 5 |
| 616884 | a | ae0a2569 4845a18d | 5 |
| 616854 | a | ea5fc581 4a85a18d | 5 |
| 616823 | 5 | 33c61a3f b105a18d | 5 |
| 6167f1 | a | c49fb8c5 63f9a18d | 5 |
| 6167be | 5 | 5a2e0cb4 5f79a18d | 5 |
| 61678a | 5 | 60009257 1479a18d | 5 |
| 616755 | a | 86314e62 eab9a18d | 5 |
| 61671f | 5 | 3defd9bb be39a18d | 5 |
| 6166e8 | a | bff7e728 c5d9a18d | 5 |
| 6166b0 | a | bda11512 4359a18d | 5 |
| 616677 | 5 | 6513b3b1 fb99a18d | 5 |
| 61663d | a | decd2468 af19a18d | 5 |
| 616602 | a | f6542b5f d019a18d | 5 |
| 6165c6 | a | dc44b49b d8e9a18d | 5 |
| 616589 | 5 | 42f500ea e469a18d | 5 |
| 61654b | a | bf2885e1 34a9a18d | 5 |
| 61650c | a | ec4c69b5 4c29a18d | 5 |

# 4 HEC AND PACKET HEADER SAMPLE DATA

This section contains examples of HECs computed for sample UAP and packet header contents (Data). The resulting 54 bit packet headers are shown in the rightmost column. Note that the UAP, Data and HEC values are in hexadecimal notation, while the header is in octal notation. The header is transmitted from left to right over the air.

| UAP | Data | HEC | Header (octal) |
|---|---|---|---|
| 00 | 123 | e1 | 770007 007070 000777 |
| 47 | 123 | 06 | 770007 007007 700000 |
| 00 | 124 | 32 | 007007 007007 007700 |
| 47 | 124 | d5 | 007007 007070 707077 |
| 00 | 125 | 5a | 707007 007007 077070 |
| 47 | 125 | bd | 707007 007070 777707 |
| 00 | 126 | e2 | 077007 007007 000777 |
| 47 | 126 | 05 | 077007 007070 700000 |
| 00 | 127 | 8a | 777007 007007 070007 |
| 47 | 127 | 6d | 777007 007070 770770 |
| 00 | 11b | 9e | 770770 007007 777007 |
| 47 | 11b | 79 | 770770 007070 077770 |
| 00 | 11c | 4d | 007770 007070 770070 |
| 47 | 11c | aa | 007770 007007 070707 |
| 00 | 11d | 25 | 707770 007070 700700 |
| 47 | 11d | c2 | 707770 007007 000077 |
| 00 | 11e | 9d | 077770 007070 777007 |
| 47 | 11e | 7a | 077770 007007 077770 |
| 00 | 11f | f5 | 777770 007070 707777 |
| 47 | 11f | 12 | 777770 007007 007000 |

# 5 CRC SAMPLE DATA

This section shows the CRC computed for a sample 10 byte payload and a UAP of 0x47.

Data:
-----

data[0] = 0x4e
data[1] = 0x01
data[2] = 0x02
data[3] = 0x03
data[4] = 0x04
data[5] = 0x05
data[6] = 0x06
data[7] = 0x07
data[8] = 0x08
data[9] = 0x09

UAP = 0x47

==> CRC = 6d d2

Codeword (hexadecimal notation):
---------------------------------

4e 01 02 03 04 05 06 07 08 09 6d d2

NB: Over the air each byte in the codeword
    is sent with the LSB first.

# 6 COMPLETE SAMPLE PACKETS

## 6.1 EXAMPLE OF DH1 PACKET

```
Packet header: (MSB...LSB)
--------------------------
LT_ADDR = 011
TYPE = 0100 (DH1)
FLOW = 0
ARQN = 1
SEQN = 0

Payload: (MSB...LSB)
--------------------
payload length: 5 bytes
logical channel = 10 (UA/I, Start L2CAP message)
flow = 1
data byte 1 = 00000001
data byte 2 = 00000010
data byte 3 = 00000011
data byte 4 = 00000100
data byte 5 = 00000101

HEC and CRC initialization: (MSB...LSB)
---------------------------------------
uap = 01000111

NO WHITENING USED




AIR DATA (LSB...MSB)

Packet header (incl HEC):
-------------------------
111111000
000000111000
000111000
000111111000000000000000

Payload (incl payload header and CRC):
--------------------------------------------------------------
01110100
10000000
01000000
11000000
00100000
10100000
1110110000110110
```

## 6.2 EXAMPLE OF DM1 PACKET

```
Packet header: (MSB...LSB)
--------------------------
LT_ADDR = 011
TYPE = 0011 (DM1)
FLOW = 0
ARQN = 1
SEQN = 0

Payload: (MSB...LSB)
--------------------
payload length: 5 bytes
logical channel = 10 (UA/I, Start L2CAP message)
flow = 1
data byte 1 = 00000001
data byte 2 = 00000010
data byte 3 = 00000011
data byte 4 = 00000100
data byte 5 = 00000101

HEC and CRC initialization: (MSB...LSB)
---------------------------------------
uap = 01000111

NO WHITENING USED




AIR DATA (LSB...MSB)

Packet header (incl HEC):
-------------------------
111111000
111111000000
000111000
111000000111111111111000

Payload (incl payload header, FEC23, CRC and 6 padded zeros):
-------------------------------------------------------------
0111010010 11001
0000000100 01011
0000110000 11110
0000100000 00111
1010000011 01100
1011000011 00010
0110000000 10001
```

# 7 SIMPLE PAIRING SAMPLE DATA

This section provides sample data for the Simple Pairing cryptographic functions (f1, f2, f3, g and the ECDH calculations).

## 7.1 P-192 SAMPLE DATA

In each data set, the bytes are ordered from least significant on the right to most significant on the left.

### 7.1.1 Data Set 1

```
Private A:   07915f86918ddc27005df1d6cf0c142b625ed2eff4a518ff
Private B:   1e636ca790b50f68f15d8dbe86244e309211d635de00e16d
Public A(x): 15207009984421a6586f9fc3fe7e4329d2809ea51125f8ed
Public A(y): b09d42b81bc5bd009f79e4b59dbbaa857fca856fb9f7ea25
DHKey:       fb3ba2012c7e62466e486e229290175b4afebc13fdccee46
```

### 7.1.2 Data Set 2

```
Private A:   52ec1ca6e0ec973c29065c3ca10be80057243002f09bb43e
Private B:   57231203533e9efe18cc622fd0e34c6a29c6e0fa3ab3bc53
Public A(x): 45571f027e0d690795d61560804da5de789a48f94ab4b07e
Public A(y): 0220016e8a6bce74b45ffec1e664aaa0273b7cbd907a8e2b
DHKey:       a20a34b5497332aa7a76ab135cc0c168333be309d463c0c0
```

### 7.1.3 Data Set 3

```
Private A:   00a0df08eaf51e6e7be519d67c6749ea3f4517cdd2e9e821
Private B:   2bf5e0d1699d50ca5025e8e2d9b13244b4d322a328be1821
Public A(x): 2ed35b430fa45f9d329186d754eeeb0495f0f653127f613d
Public A(y): 27e08db74e424395052ddae7e3d5a8fecb52a8039b735b73
DHKey:       3b3986ba70790762f282a12a6d3bcae7a2ca01e25b87724e
```

### 7.1.4 Data Set 4

```
Private A:   030a4af66e1a4d590a83e0284fca5cdf83292b84f4c71168
Private B:   12448b5c69ecd10c0471060f2bf86345c5e83c03d16bae2c
Public A(x): f24a6899218fa912e7e4a8ba9357cb8182958f9fa42c968c
Public A(y): 7c0b8a9ebe6ea92e968c3a65f9f1a9716fe826ad88c97032
DHKey:       4a78f83fba757c35f94abea43e92effdd2bc700723c61939
```

### 7.1.5 Data Set 5

```
Private A:   604df406c649cb460be16244589a40895c0db7367dc11a2f
Private B:   526c2327303cd505b9cf0c012471902bb9e842ce32b0addc
Public A(x): cbe3c629aceb41b73d475a79fbfe8c08cdc80ceec00ee7c9
```

```
Public A(y): f9f70f7ae42abda4f33af56f7f6aa383354e453fa1a2bd18
DHKey:       64d4fe35567e6ea0ca31f947e1533a635436d4870ce88c45
```

### 7.1.6 Data Set 6

```
Private A:   1a2c582a09852979eb2cee18fb0befb9a55a6d06f6a8fad3
Private B:   243778916920d68df535955bc1a3cccd5811133a8205ae41
Public A(x): eca2d8d30bbef3ba8b7d591fdb98064a6c7b870cdcebe67c
Public A(y): 2e4163a44f3ae26e70dae86f1bf786e1a5db5562a8ed9fee
DHKey:       6433b36a7e9341940e78a63e31b3cf023282f7f1e3bf83bd
```

### 7.1.7 Data Set 7

```
Private A:   0f494dd08b493edb07228058a9f30797ff147a5a2adef9b3
Private B:   2da4cd46d9e06e81b1542503f2da89372e927877becec1be
Public A(x): 9f56a8aa27346d66652a546abacc7d69c17fd66e0853989f
Public A(y): d7234c1464882250df7bbe67e0fa22aae475dc58af0c4210
DHKey:       c67beda9baf3c96a30616bf87a7d0ae704bc969e5cad354b
```

### 7.1.8 Data Set 8

```
Private A:   7381d2bc6ddecb65126564cb1af6ca1985d19fb57f0fff16
Private B:   18e276beff75adc3d520badb3806822e1c820f1064447848
Public A(x): 61c7f3c6f9e09f41423dce889de1973d346f2505a5a3b19b
Public A(y): 919972ff4cd6aed8a4821e3adc358b41f7be07ede20137df
DHKey:       6931496eef2fcfb03e0b1eef515dd4e1b0115b8b241b0b84
```

### 7.1.9 Data Set 9

```
Private A:   41c7b484ddc37ef6b7952c379f87593789dac6e4f3d8d8e6
Private B:   33e4eaa77f78216e0e99a9b200f81d2ca20dc74ad62d9b78
Public A(x): 9f09c773adb8e7b66b5d986cd15b143341a66d824113c15f
Public A(y): d2000a91738217ab8070a76c5f96c03de317dfab774f4837
DHKey:       a518f3826bb5fa3d5bc37da4217296d5b6af51e5445c6625
```

### 7.1.10 Data Set 10

```
Private A:   703cf5ee9c075f7726d0bb36d131c664f5534a6e6305d631
Private B:   757291c620a0e7e9dd13ce09ceb729c0ce1980e64d569b5f
Public A(x): fa2b96d382cf894aeeb0bd985f3891e655a6315cd5060d03
Public A(y): f7e8206d05c7255300cc56c88448158c497f2df596add7a2
DHKey:       12a3343bb453bb5408da42d20c2d0fcc18ff078f56d9c68c
```

## 7.2 HASH FUNCTIONS SAMPLE DATA

In each data set, the bytes are ordered from least significant on the right to most significant on the left.

### 7.2.1 f1()

```
Set 1a
          U:       15207009984421a6586f9fc3fe7e4329d2809ea51125f8ed
          V:       356b31938421fbbf2fb331c89fd588a69367e9a833f56812
          X:       d5cb8454d177733effffb2ec712baeab
          Z:       00
          output:  1bdc955a9d542ffc9f9e670cdf665010

Set 1b
          U:      15207009984421a6586f9fc3fe7e4329d2809ea51125f8ed
          V:      356b31938421fbbf2fb331c89fd588a69367e9a833f56812
          X:      d5cb8454d177733effffb2ec712baeab
          Z:      80
          output: 611325ebcb6e5269b868113306095fa6

Set 1c
          U:      15207009984421a6586f9fc3fe7e4329d2809ea51125f8ed
          V:      356b31938421fbbf2fb331c89fd588a69367e9a833f56812
          X:      d5cb8454d177733effffb2ec712baeab
          Z:      81
          output: b68df39fd8a406b06a6c517d3666cf91

Set 2a (swapped U and V inputs compared with set 1)
          U:       356b31938421fbbf2fb331c89fd588a69367e9a833f56812
          V:       15207009984421a6586f9fc3fe7e4329d2809ea51125f8ed
          X:       d5cb8454d177733effffb2ec712baeab
          Z:       00
          output:  f4e1ec4b88f305e81477627b1643a927

Set 2b (swapped U and V inputs compared with set 1)
          U:      356b31938421fbbf2fb331c89fd588a69367e9a833f56812
          V:      15207009984421a6586f9fc3fe7e4329d2809ea51125f8ed
          X:      d5cb8454d177733effffb2ec712baeab
          Z:      80
          output: ac6aa7cfa96ae99dd3a74225adb068ae

Set 2c (swapped U and V inputs compared with set 1)
          U:      356b31938421fbbf2fb331c89fd588a69367e9a833f56812
          V:      15207009984421a6586f9fc3fe7e4329d2809ea51125f8ed
          X:      d5cb8454d177733effffb2ec712baeab
          Z:      81
          output: 5ad4721258aa1fa06082edad980d0cc5

Set 3a (U and V set to same value as U in set 1)
          U:       15207009984421a6586f9fc3fe7e4329d2809ea51125f8ed
          V:       15207009984421a6586f9fc3fe7e4329d2809ea51125f8ed
          X:       d5cb8454d177733effffb2ec712baeab
          Z:       00
          output:  49125fc1e8cdc615826c15e5d23ede41
```

```
Set 3b (U and V set to same value as V in set 1)
          U:       356b31938421fbbf2fb331c89fd588a69367e9a833f56812
          V:       356b31938421fbbf2fb331c89fd588a69367e9a833f56812
          X:       d5cb8454d177733effffb2ec712baeab
          Z:       80
          output:  159f204c520565175c2b9c523acad2eb

Set 3c (U and V set to same value as V in set 1)
          U:       356b31938421fbbf2fb331c89fd588a69367e9a833f56812
          V:       356b31938421fbbf2fb331c89fd588a69367e9a833f56812
          X:       d5cb8454d177733effffb2ec712baeab
          Z:       81
          output:  9a162ff9a8235e5b12539ba0ff9179da
```

### 7.2.2 g()

```
Set 1
          U:15207009984421a6586f9fc3fe7e4329d2809ea51125f8ed
          V:356b31938421fbbf2fb331c89fd588a69367e9a833f56812
          X:        d5cb8454d177733effffb2ec712baeab
          Y:        a6e8e7cc25a75f6e216583f7ff3dc4cf
          output:             52146a1e
          output (decimal):   1377069598
          6 digits (decimal): 069598
```

### 7.2.3 f2()

```
Set 1
          W:      fb3ba2012c7e62466e486e229290175b4afebc13fdccee46
          N1:     d5cb8454d177733effffb2ec712baeab
          N2:     a6e8e7cc25a75f6e216583f7ff3dc4cf
          keyID:  62746c6b
          A1:     56123737bfce
          A2:     a713702dcfc1
          output: c234c1198f3b520186ab92a2f874934e
```

### 7.2.4 f3()

```
Set 1
          W:      fb3ba2012c7e62466e486e229290175b4afebc13fdccee46
          N1:     d5cb8454d177733effffb2ec712baeab
          N2:     a6e8e7cc25a75f6e216583f7ff3dc4cf
          R:      12a3343bb453bb5408da42d20c2d0fc8
          IOcap:  000000
          A1:     56123737bfce
          A2:     a713702dcfc1
          output: 5e6a346b8add7ee80e7ec0c2461b1509
```

```
Set 2
          W:      fb3ba2012c7e62466e486e229290175b4afebc13fdccee46
          N1:     d5cb8454d177733effffb2ec712baeab
          N2:     a6e8e7cc25a75f6e216583f7ff3dc4cf
          R:      12a3343bb453bb5408da42d20c2d0fc8
          IOcap:  000001
          A1:     56123737bfce
          A2:     a713702dcfc1
          output: 7840e5445a13e3ce6e48a2decbe51482

Set 3
          W:      fb3ba2012c7e62466e486e229290175b4afebc13fdccee46
          N1:     d5cb8454d177733effffb2ec712baeab
          N2:     a6e8e7cc25a75f6e216583f7ff3dc4cf
          R:      12a3343bb453bb5408da42d20c2d0fc8
          IOcap:  000002
          A1:     56123737bfce
          A2:     a713702dcfc1
          output: da9afb5c6c9dbe0af4722b532520c4b3


Set 4
          W:      fb3ba2012c7e62466e486e229290175b4afebc13fdccee46
          N1:     d5cb8454d177733effffb2ec712baeab
          N2:     a6e8e7cc25a75f6e216583f7ff3dc4cf
          R:      12a3343bb453bb5408da42d20c2d0fc8
          IOcap:  000003
          A1:     56123737bfce
          A2:     a713702dcfc1
          output: 2c0f220c50075285852e01bcee4b5f90

Set 5
          W:      fb3ba2012c7e62466e486e229290175b4afebc13fdccee46
          N1:     d5cb8454d177733effffb2ec712baeab
          N2:     a6e8e7cc25a75f6e216583f7ff3dc4cf
          R:      12a3343bb453bb5408da42d20c2d0fc8
          IOcap:  000100
          A1:     56123737bfce
          A2:     a713702dcfc1
          output: 0a096af0fa61dce0933987febe95fc7d

Set 6
          W:      fb3ba2012c7e62466e486e229290175b4afebc13fdccee46
          N1:     d5cb8454d177733effffb2ec712baeab
          N2:     a6e8e7cc25a75f6e216583f7ff3dc4cf
          R:      12a3343bb453bb5408da42d20c2d0fc8
          IOcap:  000101
          A1:     56123737bfce
```

```
          A2:     a713702dcfc1
          output: 49b8d74007888e770e1a49d6810069b9

Set 7
          W:      fb3ba2012c7e62466e486e229290175b4afebc13fdccee46
          N1:     d5cb8454d177733effffb2ec712baeab
          N2:     a6e8e7cc25a75f6e216583f7ff3dc4cf
          R:      12a3343bb453bb5408da42d20c2d0fc8
          IOcap:  000102
          A1:     56123737bfce
          A2:     a713702dcfc1
          output: 309cd0327dec2514894a0c88b101a436

Set 8
          W:      fb3ba2012c7e62466e486e229290175b4afebc13fdccee46
          N1:     d5cb8454d177733effffb2ec712baeab
          N2:     a6e8e7cc25a75f6e216583f7ff3dc4cf
          R:      12a3343bb453bb5408da42d20c2d0fc8
          IOcap:  000103
          A1:     56123737bfce
          A2:     a713702dcfc1
          output: 4512274ba875b156c2187e2061b90434

Set 9 (same as set 1 with N1 and N2 swapped and A1 and A2 swapped)
          W:      fb3ba2012c7e62466e486e229290175b4afebc13fdccee46
          N1:     a6e8e7cc25a75f6e216583f7ff3dc4cf
          N2:     d5cb8454d177733effffb2ec712baeab
          R:      12a3343bb453bb5408da42d20c2d0fc8
          IOcap:  000000
          A1:     a713702dcfc1
          A2:     56123737bfce
          output: 8d56dc59e70855f563b5e85e42d5964e

Set 10 (same as set 2 with N1 and N2 swapped and A1 and A2 swapped)
          W:      fb3ba2012c7e62466e486e229290175b4afebc13fdccee46
          N1:     a6e8e7cc25a75f6e216583f7ff3dc4cf
          N2:     d5cb8454d177733effffb2ec712baeab
          R:      12a3343bb453bb5408da42d20c2d0fc8
          IOcap:  000001
          A1:     a713702dcfc1
          A2:     56123737bfce
          output: c92fdacbf0ce931e9c4087a9dfb7bc0b

Set 11 (same as set 3 with N1 and N2 swapped and A1 and A2 swapped)
          W:      fb3ba2012c7e62466e486e229290175b4afebc13fdccee46
          N1:     a6e8e7cc25a75f6e216583f7ff3dc4cf
          N2:     d5cb8454d177733effffb2ec712baeab
          R:      12a3343bb453bb5408da42d20c2d0fc8
          IOcap:  000002
```

```
          A1:     a713702dcfc1
          A2:     56123737bfce
          output: 52ac910200dc34285bbbf2144883c498


Set 12 (same as set 4 with N1 and N2 swapped and A1 and A2 swapped)
          W:      fb3ba2012c7e62466e486e229290175b4afebc13fdccee46
          N1:     a6e8e7cc25a75f6e216583f7ff3dc4cf
          N2:     d5cb8454d177733effffb2ec712baeab
          R:      12a3343bb453bb5408da42d20c2d0fc8
          IOcap:  000003
          A1:     a713702dcfc1
          A2:     56123737bfce
          output: c419d677e0d426e6bb36de5fa54c5041

Set 13 (same as set 5 with N1 and N2 swapped and A1 and A2 swapped)
          W:      fb3ba2012c7e62466e486e229290175b4afebc13fdccee46
          N1:     a6e8e7cc25a75f6e216583f7ff3dc4cf
          N2:     d5cb8454d177733effffb2ec712baeab
          R:      12a3343bb453bb5408da42d20c2d0fc8
          IOcap:  000100
          A1:     a713702dcfc1
          A2:     56123737bfce
          output: fb0e1f9f7c623c1bf2675fcff1551137

Set 14 (same as set 6 with N1 and N2 swapped and A1 and A2 swapped)
          W:      fb3ba2012c7e62466e486e229290175b4afebc13fdccee46
          N1:     a6e8e7cc25a75f6e216583f7ff3dc4cf
          N2:     d5cb8454d177733effffb2ec712baeab
          R:      12a3343bb453bb5408da42d20c2d0fc8
          IOcap:  000101
          A1:     a713702dcfc1
          A2:     56123737bfce
          output: 16c7be68184f1170fbbb2bef5a9c515d

Set 15 (same as set 7 with N1 and N2 swapped and A1 and A2 swapped)
          W:      fb3ba2012c7e62466e486e229290175b4afebc13fdccee46
          N1:     a6e8e7cc25a75f6e216583f7ff3dc4cf
          N2:     d5cb8454d177733effffb2ec712baeab
          R:      12a3343bb453bb5408da42d20c2d0fc8
          IOcap:  000102
          A1:     a713702dcfc1
          A2:     56123737bfce
          output: 24849f33d3ac05fef9034c18d9adb310

Set 16 (same as set 8 with N1 and N2 swapped and A1 and A2 swapped)
          W:      fb3ba2012c7e62466e486e229290175b4afebc13fdccee46
          N1:     a6e8e7cc25a75f6e216583f7ff3dc4cf
          N2:     d5cb8454d177733effffb2ec712baeab
```

```
          R:      12a3343bb453bb5408da42d20c2d0fc8
          IOcap:  000103
          A1:     a713702dcfc1
          A2:     56123737bfce
          output: e0f484bb0b071483285903e85094046b


Set 17
          W:      fb3ba2012c7e62466e486e229290175b4afebc13fdccee46
          N1:     d5cb8454d177733effffb2ec712baeab
          N2:     a6e8e7cc25a75f6e216583f7ff3dc4cf
          R:      12a3343bb453bb5408da42d20c2d0fc8
          IOcap:  010000
          A1:     56123737bfce
          A2:     a713702dcfc1
          output: 4bf22677415ed90aceb21873c71c1884

Set 18
          W:      fb3ba2012c7e62466e486e229290175b4afebc13fdccee46
          N1:     d5cb8454d177733effffb2ec712baeab
          N2:     a6e8e7cc25a75f6e216583f7ff3dc4cf
          R:      12a3343bb453bb5408da42d20c2d0fc8
          IOcap:  010001
          A1:     56123737bfce
          A2:     a713702dcfc1
          output: 0d4b97992eb570efb369cfe45e1681b5

Set 19
          W:      fb3ba2012c7e62466e486e229290175b4afebc13fdccee46
          N1:     d5cb8454d177733effffb2ec712baeab
          N2:     a6e8e7cc25a75f6e216583f7ff3dc4cf
          R:      12a3343bb453bb5408da42d20c2d0fc8
          IOcap:  010002
          A1:     56123737bfce
          A2:     a713702dcfc1
          output: 0f0906bbfa75e95c471e97c4211b2741


Set 20
          W:      fb3ba2012c7e62466e486e229290175b4afebc13fdccee46
          N1:     d5cb8454d177733effffb2ec712baeab
          N2:     a6e8e7cc25a75f6e216583f7ff3dc4cf
          R:      12a3343bb453bb5408da42d20c2d0fc8
          IOcap:  010003
          A1:     56123737bfce
          A2:     a713702dcfc1
          output: 88f1f60ce1ff4bf8aa08a170dd061d4e

Set 21
```

```
          W:      fb3ba2012c7e62466e486e229290175b4afebc13fdccee46
          N1:     d5cb8454d177733effffb2ec712baeab
          N2:     a6e8e7cc25a75f6e216583f7ff3dc4cf
          R:      12a3343bb453bb5408da42d20c2d0fc8
          IOcap:  010100
          A1:     56123737bfce
          A2:     a713702dcfc1
          output: 940f88f25317c358d9bd2f146778e36b

Set 22
          W:      fb3ba2012c7e62466e486e229290175b4afebc13fdccee46
          N1:     d5cb8454d177733effffb2ec712baeab
          N2:     a6e8e7cc25a75f6e216583f7ff3dc4cf
          R:      12a3343bb453bb5408da42d20c2d0fc8
          IOcap:  010101
          A1:     56123737bfce
          A2:     a713702dcfc1
          output: 591396355ac4dc72be05a15e718ec945

Set 23
          W:      fb3ba2012c7e62466e486e229290175b4afebc13fdccee46
          N1:     d5cb8454d177733effffb2ec712baeab
          N2:     a6e8e7cc25a75f6e216583f7ff3dc4cf
          R:      12a3343bb453bb5408da42d20c2d0fc8
          IOcap:  010102
          A1:     56123737bfce
          A2:     a713702dcfc1
          output: a3dc055f656abb1d6e11d3f37340189a

Set 24
          W:      fb3ba2012c7e62466e486e229290175b4afebc13fdccee46
          N1:     d5cb8454d177733effffb2ec712baeab
          N2:     a6e8e7cc25a75f6e216583f7ff3dc4cf
          R:      12a3343bb453bb5408da42d20c2d0fc8
          IOcap:  010103
          A1:     56123737bfce
          A2:     a713702dcfc1
          output: fd5412a22ba5dd893852608f8ab0c934

Set 25 (same as set 1 with N1 and N2 swapped and A1 and A2 swapped)
          W:      fb3ba2012c7e62466e486e229290175b4afebc13fdccee46
          N1:     a6e8e7cc25a75f6e216583f7ff3dc4cf
          N2:     d5cb8454d177733effffb2ec712baeab
          R:      12a3343bb453bb5408da42d20c2d0fc8
          IOcap:  010000
          A1:     a713702dcfc1
          A2:     56123737bfce
          output: 2a742039c5fd6c6faafce17b619ac56f
```

```
Set 26 (same as set 2 with N1 and N2 swapped and A1 and A2 swapped)
          W:      fb3ba2012c7e62466e486e229290175b4afebc13fdccee46
          N1:     a6e8e7cc25a75f6e216583f7ff3dc4cf
          N2:     d5cb8454d177733effffb2ec712baeab
          R:      12a3343bb453bb5408da42d20c2d0fc8
          IOcap:  010001
          A1:     a713702dcfc1
          A2:     56123737bfce
          output: a60d89efb52db7905179a6c63b8f212a

Set 27 (same as set 3 with N1 and N2 swapped and A1 and A2 swapped)
          W:      fb3ba2012c7e62466e486e229290175b4afebc13fdccee46
          N1:     a6e8e7cc25a75f6e216583f7ff3dc4cf
          N2:     d5cb8454d177733effffb2ec712baeab
          R:      12a3343bb453bb5408da42d20c2d0fc8
          IOcap:  010002
          A1:     a713702dcfc1
          A2:     56123737bfce
          output: cb02f803d755fd936f0a832f4ee9fd4a


Set 28 (same as set 4 with N1 and N2 swapped and A1 and A2 swapped)
          W:      fb3ba2012c7e62466e486e229290175b4afebc13fdccee46
          N1:     a6e8e7cc25a75f6e216583f7ff3dc4cf
          N2:     d5cb8454d177733effffb2ec712baeab
          R:      12a3343bb453bb5408da42d20c2d0fc8
          IOcap:  010003
          A1:     a713702dcfc1
          A2:     56123737bfce
          output: 00786c04a24561485aaf22808871b874

Set 29 (same as set 5 with N1 and N2 swapped and A1 and A2 swapped)
          W:      fb3ba2012c7e62466e486e229290175b4afebc13fdccee46
          N1:     a6e8e7cc25a75f6e216583f7ff3dc4cf
          N2:     d5cb8454d177733effffb2ec712baeab
          R:      12a3343bb453bb5408da42d20c2d0fc8
          IOcap:  010100
          A1:     a713702dcfc1
          A2:     56123737bfce
          output: 2a58ef2d99281d472a88027423f8215f

Set 30 (same as set 6 with N1 and N2 swapped and A1 and A2 swapped)
          W:      fb3ba2012c7e62466e486e229290175b4afebc13fdccee46
          N1:     a6e8e7cc25a75f6e216583f7ff3dc4cf
          N2:     d5cb8454d177733effffb2ec712baeab
          R:      12a3343bb453bb5408da42d20c2d0fc8
          IOcap:  010101
          A1:     a713702dcfc1
          A2:     56123737bfce
```

```
          output: ff7ab3752a144232f2cbbcbf979f5517

Set 31 (same as set 7 with N1 and N2 swapped and A1 and A2 swapped)
          W:      fb3ba2012c7e62466e486e229290175b4afebc13fdccee46
          N1:     a6e8e7cc25a75f6e216583f7ff3dc4cf
          N2:     d5cb8454d177733effffb2ec712baeab
          R:      12a3343bb453bb5408da42d20c2d0fc8
          IOcap:  010102
          A1:     a713702dcfc1
          A2:     56123737bfce
          output: 9d7fccef23625b1cc684fbf3f8b0e182

Set 32 (same as set 8 with N1 and N2 swapped and A1 and A2 swapped)
          W:      fb3ba2012c7e62466e486e229290175b4afebc13fdccee46
          N1:     a6e8e7cc25a75f6e216583f7ff3dc4cf
          N2:     d5cb8454d177733effffb2ec712baeab
          R:      12a3343bb453bb5408da42d20c2d0fc8
          IOcap:  010103
          A1:     a713702dcfc1
          A2:     56123737bfce
          output: 864f87e26dece4dfdde30ade1e463d4f
```

### 7.2.5 h2()

Each element in this section except for 'L' is a multi-octet integer arranged with the least significant octet on the right and the most significant octet on the left.

```
Set 1 (Generic AMP initial creation)
    W:      c234c1198f3b520186ab92a2f874934ec234c1198f3b520186ab92a2f874934e
    keyID:  67616d70 ('gamp')
    L:      32
    output: 7E010832407F59D51CB473FB0D3355899C077DD052D0F5F03F0FA4B964097A04

Set 2 (802.11)
    W:      7E010832407F59D51CB473FB0D3355899C077DD052D0F5F03F0FA4B964097A04
    keyID:  38303262 ('802b')
    L:      32
    output: D660DC08E542F1DCC32CDC2000B548F06BC1D7EB0F227DD809B50D9F1945C68E

Set 3 (Generic AMP refresh after 802.11)
    W:      7E010832407F59D51CB473FB0D3355899C077DD052D0F5F03F0FA4B964097A04
    keyID:  67616d70 ('gamp')
    L:      32
    output: AA8247E5DBAFC80F908C1C2ECA2F73E86D41DF754355994B220D5817040991B2

Set 4 (ECMA-368)
    W:      AA8247E5DBAFC80F908C1C2ECA2F73E86D41DF754355994B220D5817040991B2
    keyID:  65636d61 ('ecma')
```

*Sample Data*

```
    L:      16
    output: 89286411644B33DC378A93A1B1506AFF

Set 5 (Generic AMP refresh after ECMA-368)
    W:      AA8247E5DBAFC80F908C1C2ECA2F73E86D41DF754355994B220D5817040991B2
    keyID:  67616d70 ('gamp')
    L:      32
    output: 925CAE4683A2CCF105ED141F5370AB501448E83D2E43F38573120360B100F27D
```

# 8 WHITENING SEQUENCE SAMPLE DATA

This section shows the output of the whitening sequence generator.

| Whitening Sequence (=D7) | Whitening LFSR D7.....D0 |
|---|---|
| 1 | 1111111 |
| 1 | 1101111 |
| 1 | 1001111 |
| 0 | 0001111 |
| 0 | 0011110 |
| 0 | 0111100 |
| 1 | 1111000 |
| 1 | 1100001 |
| 1 | 1010011 |
| 0 | 0110111 |
| 1 | 1101110 |
| 1 | 1001101 |
| 0 | 0001011 |
| 0 | 0010110 |
| 0 | 0101100 |
| 1 | 1011000 |
| 0 | 0100001 |
| 1 | 1000010 |
| 0 | 0010101 |
| 0 | 0101010 |
| 1 | 1010100 |
| 0 | 0111001 |
| 1 | 1110010 |
| 1 | 1110101 |
| 1 | 1111011 |
| 1 | 1100111 |
| 1 | 1011111 |
| 0 | 0101111 |
| 1 | 1011110 |
| 0 | 0101101 |
| 1 | 1011010 |
| 0 | 0100101 |
| 1 | 1001010 |
| 0 | 0000101 |
| 0 | 0001010 |
| 0 | 0010100 |
| 0 | 0101000 |
| 1 | 1010000 |
| 0 | 0110001 |
| 1 | 1100010 |
| 1 | 1010101 |
| 0 | 0111011 |
| 1 | 1110110 |

*Sample Data*

| | |
|---|---|
| 1 | 1111101 |
| 1 | 1101011 |
| 1 | 1000111 |
| 0 | 0011111 |
| 0 | 0111110 |
| 1 | 1111100 |
| 1 | 1101001 |
| 1 | 1000011 |
| 0 | 0010111 |
| 0 | 0101110 |
| 1 | 1011100 |
| 0 | 0101001 |
| 1 | 1010010 |
| 0 | 0110101 |
| 1 | 1101010 |
| 1 | 1000101 |
| 0 | 0011011 |
| 0 | 0110110 |
| 1 | 1101100 |
| 1 | 1001001 |
| 0 | 0000011 |
| 0 | 0000110 |
| 0 | 0001100 |
| 0 | 0011000 |
| 0 | 0110000 |
| 1 | 1100000 |
| 1 | 1010001 |
| 0 | 0110011 |
| 1 | 1100110 |
| 1 | 1011101 |
| 0 | 0101011 |
| 1 | 1010110 |
| 0 | 0111101 |
| 1 | 1111010 |
| 1 | 1100101 |
| 1 | 1011011 |
| 0 | 0100111 |
| 1 | 1001110 |
| 0 | 0001101 |
| 0 | 0011010 |
| 0 | 0110100 |
| 1 | 1101000 |
| 1 | 1000001 |
| 0 | 0010011 |
| 0 | 0100110 |
| 1 | 1001100 |
| 0 | 0001001 |
| 0 | 0010010 |
| 0 | 0100100 |
| 1 | 1001000 |
| 0 | 0000001 |
| 0 | 0000010 |

*Sample Data*

| | |
|---|---|
| 0 | 0000100 |
| 0 | 0001000 |
| 0 | 0010000 |
| 0 | 0100000 |
| 1 | 1000000 |
| 0 | 0010001 |
| 0 | 0100010 |
| 1 | 1000100 |
| 0 | 0011001 |
| 0 | 0110010 |
| 1 | 1100100 |
| 1 | 1011001 |
| 0 | 0100011 |
| 1 | 1000110 |
| 0 | 0011101 |
| 0 | 0111010 |
| 1 | 1110100 |
| 1 | 1111001 |
| 1 | 1100011 |
| 1 | 1010111 |
| 0 | 0111111 |
| 1 | 1111110 |
| 1 | 1101101 |
| 1 | 1001011 |
| 0 | 0000111 |
| 0 | 0001110 |
| 0 | 0011100 |
| 0 | 0111000 |
| 1 | 1110000 |
| 1 | 1110001 |
| 1 | 1110011 |
| 1 | 1110111 |
| 1 | 1111111 |

# 9 FEC SAMPLE DATA

```
================================================
 Rate 2/3 FEC -- (15,10) Shortened Hamming Code
================================================
```

Data is in hexadecimal notation, the codewords are in binary notation. The codeword bits are sent from left to right over the air interface. The space in the codeword indicates the start of parity bits.

| Data: | Codeword: |
|---|---|
| 0x001 | 1000000000 11010 |
| 0x002 | 0100000000 01101 |
| 0x004 | 0010000000 11100 |
| 0x008 | 0001000000 01110 |
| 0x010 | 0000100000 00111 |
| 0x020 | 0000010000 11001 |
| 0x040 | 0000001000 10110 |
| 0x080 | 0000000100 01011 |
| 0x100 | 0000000010 11111 |
| 0x200 | 0000000001 10101 |

# 10 ENCRYPTION KEY SAMPLE DATA

Explanation:

For the Section 10.1 through Section 10.5, the hexadecimal sample data is written with the least significant byte at the leftmost position and the most significant byte at the rightmost position. Within each byte, the *least significant bit* (LSB) is at the rightmost position and the *most significant bit* (MSB) is at the leftmost position Thus, a line reading:

aco: 48afcdd4bd40fef76693b113

means aco[0]=0x48, ac[1]=0xaf, ..., aco[11]=0x13. The LSB of aco[11] is '1' and the MSB of aco[11] is '0'.

Key [ i]: denotes the ith sub-key in Ar or A'r;
round r: denotes the input to the rth round;
added ->: denotes the input to round 3 in A'r after adding original input (of round 1).

## 10.1 FOUR TESTS OF E1

```
rand    :00000000000000000000000000000000
address :000000000000
key     :00000000000000000000000000000000
round  1:00000000000000000000000000000000
Key [ 1]:00000000000000000000000000000000
Key [ 2]:4697b1baa3b7100ac537b3c95a28ac64
round  2:78d19f9307d2476a523ec7a8a026042a
Key [ 3]:ecabaac66795580df89af66e66dc053d
Key [ 4]:8ac3d8896ae9364943bfebd4969b68a0
round  3:600265247668dda0e81c07bbb30ed503
Key [ 5]:5d57921fd5715cbb22c1be7bbc996394
Key [ 6]:2a61b8343219fdfb1740e6511d41448f
round  4:d7552ef7cc9dbde568d80c2215bc4277
Key [ 7]:dd0480dee731d67f01a2f739da6f23ca
Key [ 8]:3ad01cd1303e12a1cd0fe0a8af82592c
round  5:fb06bef32b52ab8f2a4f2b6ef7f6d0cd
Key [ 9]:7dadb2efc287ce75061302904f2e7233
Key [10]:c08dcfa981e2c4272f6c7a9f52e11538
round  6:b46b711ebb3cf69e847a75f0ab884bdd
Key [11]:fc2042c708e409555e8c147660ffdfd7
Key [12]:fa0b21001af9a6b9e89e624cd99150d2
round  7:c585f308ff19404294f06b292e978994
Key [13]:18b40784ea5ba4c80ecb48694b4e9c35
Key [14]:454d54e5253c0c4a8b3fcca7db6baef4
round  8:2665fadb13acf952bf74b4ab12264b9f
Key [15]:2df37c6d9db52674f29353b0f011ed83
Key [16]:b60316733b1e8e70bd861b477e2456f1
Key [17]:884697b1baa3b7100ac537b3c95a28ac
```

*Sample Data*

```
round  1:158ffe43352085e8a5ec7a88e1ff2ba8
Key [ 1]:e9e5dfc1b3a79583e9e5dfc1b3a79583
Key [ 2]:7595bf57e0632c59f435c16697d4c864
round  2:0b5cc75febcdf7827ca29ec0901b6b5b
Key [ 3]:e31b96afcc75d286ef0ae257cbbc05b7
Key [ 4]:0d2a27b471bc0108c6263aff9d9b3b6b
round  3:e4278526c8429211f7f2f0016220aef4
added ->:f1b68365fd6217f952de6a89831fd95c
Key [ 5]:98d1eb5773cf59d75d3b17b3bc37c191
Key [ 6]:fd2b79282408ddd4ea0aa7511133336f
round  4:d0304ad18337f86040145d27aa5c8153
Key [ 7]:331227756638a41d57b0f7e071ee2a98
Key [ 8]:aa0dd8cc68b406533d0f1d64aabacf20
round  5:84db909d213bb0172b8b6aaf71bf1472
Key [ 9]:669291b0752e63f806fce76f10e119c8
Key [10]:ef8bdd46be8ee0277e9b78adef1ec154
round  6:f835f52921e903dfa762f1df5abd7f95
Key [11]:f3902eb06dc409cfd78384624964bf51
Key [12]:7d72702b21f97984a721c99b0498239d
round  7:ae6c0b4bb09f25c6a5d9788a31b605d1
Key [13]:532e60bceaf902c52a06c2c283ecfa32
Key [14]:181715e5192efb2a64129668cf5d9dd4
round  8:744a6235b86cc0b853cc9f74f6b65311
Key [15]:83017c1434342d4290e961578790f451
Key [16]:2603532f365604646ff65803795ccce5
Key [17]:882f7c907b565ea58dae1c928a0dcf41
sres    :056c0fe6
aco     :48afcdd4bd40fef76693b113
----------------------------------------
rand    :bc3f30689647c8d7c5a03ca80a91eceb
address :7ca89b233c2d
key     :159dd9f43fc3d328efba0cd8a861fa57
round  1:bc3f30689647c8d7c5a03ca80a91eceb
Key [ 1]:159dd9f43fc3d328efba0cd8a861fa57
Key [ 2]:326558b3c15551899a97790e65ff669e
round  2:3e950edf197615638cc19c09f8fedc9b
Key [ 3]:62e879b65b9f53bbfbd020c624b1d682
Key [ 4]:73415f30bac8ab61f410adc9442992db
round  3:6a7640791cb536678936c5ecd4ae5a73
Key [ 5]:5093cfa1d31c1c48acd76df030ea3c31
Key [ 6]:0b4acc2b8f1f694fc7bd91f4a70f3009
round  4:fca2c022a577e2ffb2aa007589693ec7
Key [ 7]:2ca43fc817947804ecff148d50d6f6c6
Key [ 8]:3fcd73524b533e00b7f7825bea2040a4
round  5:e97f8ea4ed1a6f4a36ffc179dc6bb563
Key [ 9]:6c67bec76ae8c8cc4d289f69436d3506
Key [10]:95ed95ee8cb97e61d75848464bffb379
round  6:38b07261d7340d028749de1773a415c7
Key [11]:ff566c1fc6b9da9ac502514550f3e9d2
Key [12]:ab5ce3f5c887d0f49b87e0d380e12f47
round  7:58241f1aed7c1c3e047d724331a0b774
Key [13]:a2cab6f95eac7d655dbe84a6cd4c47f5
```

*Sample Data*

```
Key [14]:f5caff88af0af8c42a20b5bbd2c8b460
round  8:3d1aaeff53c0910de63b9788b13c490f
Key [15]:185099c1131cf97001e2f36fda415025
Key [16]:a0ebb82676bc75e8378b189eff3f6b1d
Key [17]:cf5b348aaee27ae332b4f1bfa10289a6
round  1:2e4b417b9a2a9cfd7d8417d9a6a556eb
Key [ 1]:fe78b835f26468ab069fd3991b086fda
Key [ 2]:095c5a51c6fa6d3ac1d57fa19aa382bd
round  2:b8bca81d6bb45af9d92beadd9300f5ed
Key [ 3]:1af866df817fd9f4ec00bc704192cffc
Key [ 4]:f4a8a059c1f575f076f5fbb24bf16590
round  3:351aa16dec2c3a4787080249ed323eae
added ->:1b65e2167656d6bafa8c19904bd79445
Key [ 5]:8c9d18d9356a9954d341b4286e88ea1f
Key [ 6]:5c958d370102c9881bf753e69c7da029
round  4:2ce8fef47dda6a5bee74372e33e478a2
Key [ 7]:7eb2985c3697429fbe0da334bb51f795
Key [ 8]:af900f4b63a1138e2874bfb7c628b7b8
round  5:572787f563e1643c1c862b7555637fb4
Key [ 9]:834c8588dd8f3d4f31117a488420d69b
Key [10]:bc2b9b81c15d9a80262f3f48e9045895
round  6:16b4968c5d02853c3a43aa4cdb5f26ac
Key [11]:f08608c9e39ad3147cba61327919c958
Key [12]:2d4131decf4fa3a959084714a9e85c11
round  7:10e4120c7cccef9dd4ba4e6da8571b01
Key [13]:c934fd319c4a2b5361fa8eef05ae9572
Key [14]:4904c17aa47868e40471007cde3a97c0
round  8:f9081772498fed41b6ffd72b71fcf6c6
Key [15]:ea5e28687e97fa3f833401c86e6053ef
Key [16]:1168f58252c4ecfccafbdb3af857b9f2
Key [17]:b3440f69ef951b78b5cbd6866275301b
sres    :8d5205c5
aco     :3ed75df4abd9af638d144e94
----------------------------------------
rand    :0891caee063f5da1809577ff94ccdcfb
address :c62f19f6ce98
key     :45298d06e46bac21421ddfbed94c032b
round  1:0891caee063f5da1809577ff94ccdcfb
Key [ 1]:45298d06e46bac21421ddfbed94c032b
Key [ 2]:8f03e1e1fe1c191cad35a897bc400597
round  2:1c6ca013480a685c1b28e0317f7167e1
Key [ 3]:4f2ce3a092dde854ef496c8126a69e8e
Key [ 4]:968caee2ac6d7008c07283daec67f2f2
round  3:06b4915f5fcc1fc551a52048f0af8a26
Key [ 5]:ab0d5c31f94259a6bf85ee2d22edf56c
Key [ 6]:dfb74855c0085ce73dc17b84bfd50a92
round  4:077a92b040acc86e6e0a877db197a167
Key [ 7]:8f888952662b3db00d4e904e7ea53b5d
Key [ 8]:5e18bfcc07799b0132db88cd6042f599
round  5:7204881fb300914825fdc863e8ceadf3
Key [ 9]:bfca91ad9bd3d1a06c582b1d5512dddf
Key [10]:a88bc477e3fa1d5a59b5e6cf793c7a41
```

*Sample Data*

```
round  6:27031131d86cea2d747deb4f756143aa
Key [11]:f3cfb8dac8aea2a6a8ef95af3a2a2767
Key [12]:77beb90670c5300b03aa2b2232d3d40c
round  7:fc8c13d49149b1ce8d86f96e44a00065
Key [13]:b578373650af36a06e19fe335d726d32
Key [14]:6bcee918c7d0d24dfdf42237fcf99d53
round  8:04ef5f5a7ddf846cda0a07782fc23866
Key [15]:399f158241eb3e079f45d7b96490e7ea
Key [16]:1bcfbe98ecde2add52aa63ea79fb917a
Key [17]:ee8bc03ec08722bc2b075492873374af
round  1:d989d7a40cde7032d17b52f8117b69d5
Key [ 1]:2ecc6cc797cc41a2ab02007f6af396ae
Key [ 2]:acfaef7609c12567d537ae1cf9dc2198
round  2:8e76eb9a29b2ad5eea790db97aee37c1
Key [ 3]:079c8ff9b73d428df879906a0b87a6c8
Key [ 4]:19f2710baf403a494193d201f3a8c439
round  3:346bb7c35b2539676375aafe3af69089
added ->:edf48e675703a955b2f0fc062b71f95c
Key [ 5]:d623a6498f915cb2c8002765247b2f5a
Key [ 6]:900109093319bc30108b3d9434a77a72
round  4:fafb6c1f3ebbd2477be2da49dd923f69
Key [ 7]:e28e2ee6e72e7f4e5b5c11f10d204228
Key [ 8]:8e455cd11f8b9073a2dfa5413c7a4bc5
round  5:7c72230df588060a3cf920f9b0a08f06
Key [ 9]:28afb26e2c7a64238c41cefc16c53e74
Key [10]:d08dcafc2096395ba0d2dddd0e471f4d
round  6:55991df991db26ff00073a12baa3031d
Key [11]:fcffdcc3ad8faae091a7055b934f87c1
Key [12]:f8df082d77060252c02d91e55bd6a7d6
round  7:70ec682ff864375f63701fa4f6be5377
Key [13]:bef3706e523d708e8a44147d7508bc35
Key [14]:3e98ab283ca2422d56a56cf8b06caeb3
round  8:172f12ec933da85504b4ea5c90f8f0ea
Key [15]:87ad9625d06645d22598dd5ef811ea2c
Key [16]:8bd3db0cc8168009e5da90877e13a36f
Key [17]:0e74631d813a8351ac7039b348c41b42
sres    :00507e5f
aco     :2a5f19fbf60907e69f39ca9f
----------------------------------------
rand    :0ecd61782b4128480c05dc45542b1b8c
address :f428f0e624b3
key     :35949a914225fabad91995d226de1d92
round  1:0ecd61782b4128480c05dc45542b1b8c
Key [ 1]:35949a914225fabad91995d226de1d92
Key [ 2]:ea6b3dcccc8ee5d88de349fa5010404f
round  2:8935e2e263fbc4b9302cabdfc06bce3e
Key [ 3]:920f3a0f2543ce535d4e7f25ad80648a
Key [ 4]:ad47227edf9c6874e80ba80ebb95d2c9
round  3:b4c8b878675f184a0c72f3dab51f8f05
Key [ 5]:81a941ca7202b5e884ae8fa493ecac3d
Key [ 6]:bcde1520bee3660e86ce2f0fb78b9157
round  4:77ced9f2fc42bdd5c6312b87fc2377c5
```

*Sample Data*

```
Key [ 7]:c8eee7423d7c6efa75ecec0d2cd969d3
Key [ 8]:910b3f838a02ed441fbe863a02b4a1d0
round  5:fe28e8056f3004d60bb207e628b39cf2
Key [ 9]:56c647c1e865eb078348962ae070972d
Key [10]:883965da77ca5812d8104e2b640aec0d
round  6:1f2ba92259d9e88101518f145a33840f
Key [11]:61d4cb7e4f8868a283327806a9bd8d4d
Key [12]:9f57de3a3ff310e21dc1e696ce060304
round  7:cc9b5d0218d29037e88475152ebebb2f
Key [13]:7aa1d8adc1aeed7127ef9a18f6eb2d8e
Key [14]:b4db9da3bf865912acd14904c7f7785d
round  8:b04d352bedc02682e4a7f59d7cda1dba
Key [15]:a13d7141ef1f6c7d867e3d175467381b
Key [16]:08b2bc058e50d6141cdd566a307e1acc
Key [17]:057b2b4b4be5dc0ac49e50489b8006c9
round  1:5cfacc773bae995cd7f1b81e7c9ec7df
Key [ 1]:1e717950f5828f3930fe4a9395858815
Key [ 2]:d1623369b733d98bbc894f75866c544c
round  2:d571ffa21d9daa797b1a0a3c962fc64c
Key [ 3]:4abf27664ae364cc8a7e5bcf88214cc4
Key [ 4]:2aaedda8dc4933dd6aeaf6e5c0d5a482
round  3:e17c8e498a00f125bf654c938c23f36d
added ->:bd765a3eb1ae8a796856048df0c1bab2
Key [ 5]:bc7f8ab2d86000f47b1946cc8d7a7a2b
Key [ 6]:6b28544cb13ec6c5d98470df2cf900b7
round  4:a9727c26f2f06bd9920e83c8605dcd76
Key [ 7]:1be840d9107f2c9523f66bb19f5464a1
Key [ 8]:61d6fb1aa2f0c2b26fb2a3d6de8c177c
round  5:aeff751f146eab7e4626b2e2c9e2fb39
Key [ 9]:adabfc82570c568a233173099f23f4c2
Key [10]:b7df6b55ad266c0f1ff7452101f59101
round  6:cf412b95f454d5185e67ca671892e5bd
Key [11]:8e04a7282a2950dcbaea28f300e22de3
Key [12]:21362c114433e29bda3e4d51f803b0cf
round  7:16165722fe4e07ef88f8056b17d89567
Key [13]:710c8fd5bb3cbb5f132a7061de518bd9
Key [14]:0791de7334f4c87285809343f3ead3bd
round  8:28854cd6ad4a3c572b15490d4b81bc3f
Key [15]:4f47f0e5629a674bfcd13770eb3a3bd9
Key [16]:58a6d9a16a284cc0aead2126c79608a1
Key [17]:a564082a0a98399f43f535fd5cefad34
sres    :80e5629c
aco     :a6fe4dcde3924611d3cc6ba1


========================================
```

## 10.2 FOUR TESTS OF E21

```
rand    :00000000000000000000000000000000
address :000000000000
round  1:00000000000000000000000000000000
Key [ 1]:00000000000000000000000000000006
Key [ 2]:4697b1baa3b7100ac537b3c95a28dc94
round  2:98611307ab76bbde9a86af1ce8cad412
Key [ 3]:ecabaac66795580df89af66e665d863d
Key [ 4]:8ac3d8896ae9364943bfebd4a2a768a0
round  3:820999ad2e6618f4b578974beeedf9e7
added ->:820999ad2e6618f4b578974beeedf9e7
Key [ 5]:5d57921fd5715cbb22c1bedb1c996394
Key [ 6]:2a61b8343219fdfb1740e9541d41448f
round  4:acd6edec87581ac22dbdc64ea4ced3a2
Key [ 7]:dd0480dee731d67f01ba0f39da6f23ca
Key [ 8]:3ad01cd1303e12a18dcfe0a8af82592c
round  5:1c7798732f09fbfe25795a4a2fbc93c2
Key [ 9]:7dadb2efc287ce7b0c1302904f2e7233
Key [10]:c08dcfa981e2f4572f6c7a9f52e11538
round  6:c05b88b56aa70e9c40c79bb81cd911bd
Key [11]:fc2042c708658a555e8c147660ffdfd7
Key [12]:fa0b21002605a6b9e89e624cd99150d2
round  7:abacc71b481c84c798d1bdf3d62f7e20
Key [13]:18b407e44a5ba4c80ecb48694b4e9c35
Key [14]:454d57e8253c0c4a8b3fcca7db6baef4
round  8:e8204e1183ae85cf19edb2c86215b700
Key [15]:2d0b946d9db52674f29353b0f011ed83
Key [16]:76c316733b1e8e70bd861b477e2456f1
Key [17]:8e4697b1baa3b7100ac537b3c95a28ac
Ka      :d14ca028545ec262cee700e39b5c39ee
----------------------------------------
rand    :2dd9a550343191304013b2d7e1189d09
address :cac4364303b6
round  1:cac4364303b6cac4364303b6cac43643
Key [ 1]:2dd9a550343191304013b2d7e1189d0f
Key [ 2]:14c4335b2c43910c5dcc71d81a14242b
round  2:e169f788aad45a9011f11db5270b1277
Key [ 3]:55bfb712cba168d1a48f6e74cd9f4388
Key [ 4]:2a2b3aacca695caef2821b0fb48cc253
round  3:540f9c76652e92c44987c617035037bf
added ->:9ed3d23566e45c007fcac9a1c9146dfc
Key [ 5]:a06aab22d9a2873840429 76b4b6b00ee
Key [ 6]:c229d054bb72e8eb230e6dcdb32d16b7
round  4:83659a41675f7171ea57909dc5a79ab4
Key [ 7]:23c4812ab1905ddf77dedaed4105649a
Key [ 8]:40d87e272a7a1554ae2e85e3638cdf52
round  5:0b9382d0ed4f2fccdbb69d0db7b130a4
Key [ 9]:bdc064c6a39f6b84fe40db359f62a3c4
Key [10]:58228db841ce3cee983aa721f36aa1b9
round  6:c6ebda0f8f489792f09c189568226c1f
Key [11]:a815bacd6fa747a0d4f52883ac63ebe7
```

*Sample Data*

```
Key [12]:a9ce513b38ea006c333ecaaefcf1d0f8
round  7:75a8aba07e69c9065bcd831c40115116
Key [13]:3635e074792d4122130e5b824e52cd60
Key [14]:511bdb61bb28de72a5d794bffbf407df
round  8:57a6e279dcb764cf7dd6a749dd60c735
Key [15]:a32f5f21044b6744b6d913b13cdb4c0a
Key [16]:9722bbaeef281496ef8c23a9d41e92f4
Key [17]:807370560ad7e8a13a054a65a03b4049
Ka      :e62f8bac609139b3999aedbc9d228042
----------------------------------------
rand    :dab3cffe9d5739d1b7bf4a667ae5ee24
address :02f8fd4cd661
round  1:02f8fd4cd66102f8fd4cd66102f8fd4c
Key [ 1]:dab3cffe9d5739d1b7bf4a667ae5ee22
Key [ 2]:e315a8a65d809ec7c289e69c899fbdcc
round  2:ef85ff081b8709405e19f3e275cec7dc
Key [ 3]:df6a119bb50945fc8a3394e7216448f3
Key [ 4]:87fe86fb0d58b5dd0fb3b6b1dab51d07
round  3:aa25c21bf577d92dd97381e3e9edcc54
added ->:a81dbf5723d8dbd524bf5782ebe5c918
Key [ 5]:36cc253c506c0021c91fac9d8c469e90
Key [ 6]:d5fda00f113e303809b7f7d78a1a2b0e
round  4:9e69ce9b53caec3990894d2baed41e0d
Key [ 7]:c14b5edc10cabf16bc2a2ba4a8ae1e40
Key [ 8]:74c6131afc8dce7e11b03b1ea8610c16
round  5:a5460fa8cedca48a14fd02209e01f02e
Key [ 9]:346cfc553c6cbc9713edb55f4dcbc96c
Key [10]:bddf027cb059d58f0509f8963e9bdec6
round  6:92b33f11eadcacc5a43dd05f13d334dd
Key [11]:8eb9e040c36c4c0b4a7fd3dd354d53c4
Key [12]:c6ffecdd5e135b20879b9dfa4b34bf51
round  7:fb0541aa5e5df1a61c51aef606eb5a41
Key [13]:bf12f5a6ba08dfc4fda4bdfc68c997d9
Key [14]:37c4656b9215f3c959ea688fb64ad327
round  8:f0bbd2b94ae374346730581fc77a9c98
Key [15]:e87bb0d86bf421ea4f779a8eee3a866c
Key [16]:faa471e934fd415ae4c0113ec7f0a5ad
Key [17]:95204a80b8400e49db7cf6fd2fd40d9a
Ka      :b0376d0a9b338c2e133c32b69cb816b3
----------------------------------------
rand    :13ecad08ad63c37f8a54dc56e82f4dc1
address :9846c5ead4d9
round  1:9846c5ead4d99846c5ead4d99846c5ea
Key [ 1]:13ecad08ad63c37f8a54dc56e82f4dc7
Key [ 2]:ad04f127bed50b5e671d6510d392eaed
round  2:97374e18cdd0a6f7a5aa49d1ac875c84
Key [ 3]:57ad159e5774fa222f2f3039b9cd5101
Key [ 4]:9a1e9e1068fede02ef90496e25fd8e79
round  3:9dd3260373edd9d5f4e774826b88fd2d
added ->:0519ebe9a7c6719331d1485bf3cec2c7
Key [ 5]:378dce167db62920b0b392f7cfca316e
Key [ 6]:db4277795c87286faee6c9e9a6b71a93
```

```
round  4:40ec6563450299ac4e120d88672504d6
Key [ 7]:ec01aa2f5a8a793b36c1bb858d254380
Key [ 8]:2921a66cfa5bf74ac535424564830e98
round  5:57287bbb041bd6a56c2bd931ed410cd4
Key [ 9]:07018e45aab61b3c3726ee3d57dbd5f6
Key [10]:627381f0fa4c02b0c7d3e7dfbffc3333
round  6:66affa66a8dcd36e36bf6c3f1c6a276e
Key [11]:33b57c925bd5551999f716e138efbe79
Key [12]:a6dc7f9aa95bcc9243aebd12608f657a
round  7:450e65184fd8c72c578d5cdecb286743
Key [13]:a6a6db00fd8c72a28ea57ea542f6e102
Key [14]:dcf3377daeb2e24e61f0ad6620951c1f
round  8:e5eb180b519a4e673f21b7c4f4573f3d
Key [15]:621240b9506b462a7fa250da41844626
Key [16]:ae297810f01f43dc35756cd119ee73d6
Key [17]:b959835ec2501ad3894f8b8f1f4257f9
Ka      :5b61e83ad04d23e9d1c698851fa30447
========================================
```

## 10.3 THREE TESTS OF E22

(for K_master and overlay generation)

```
rand    :001de169248850245a5f7cc7f0d6d633
PIN     :d5a51083a04a1971f18649ea8b79311a
round  1:001de169248850245a5f7cc7f0d6d623
Key [ 1]:d5a51083a04a1971f18649ea8b79311a
Key [ 2]:7317cdbff57f9b99f9810a2525b17cc7
round  2:5f05c143347b59acae3cb002db23830f
Key [ 3]:f08bd258adf1d4ae4a54d8ccb26220b2
Key [ 4]:91046cbb4ccc43db18d6dd36ca7313eb
round  3:c8f3e3300541a25b6ac5a80c3105f3c4
added ->:c810c45921c9f27f302424cbc1dbc9e7
Key [ 5]:67fb2336f4d9f069da58d11c82f6bd95
Key [ 6]:4fed702c75bd72c0d3d8f38707134c50
round  4:bd5e0c3a97fa55b91a3bbbf306ebb978
Key [ 7]:41c947f80cdc0464c50aa89070af314c
Key [ 8]:680eecfa8daf41c7109c9a5cb1f26d75
round  5:21c1a762c3cc33e75ce8976a73983087
Key [ 9]:6e33fbd94d00ff8f72e8a7a0d2cebc4c
Key [10]:f4d726054c6b948add99fabb5733ddc3
round  6:56d0df484345582f6b574a449ba155eb
Key [11]:4eda2425546a24cac790f49ef2453b53
Key [12]:cf2213624ed1510408a5a3e00b7333df
round  7:120cf9963fe9ff22993f7fdf9600d9b8
Key [13]:d04b1a25b0b8fec946d5ecfa626d04c9
Key [14]:01e5611b0f0e140bdb64585fd3ae5269
round  8:a6337400ad8cb47fefb91332f5cb2713
Key [15]:f15b2dc433f534f61bf718770a3698b1
Key [16]:f990d0273d8ea2b9e0b45917a781c720
Key [17]:f41b3cc13d4301297bb6bdfcb3e5a1dd
Ka      :539e4f2732e5ae2de1e0401f0813bd0d
```

*Sample Data*

```
---------------------------------------
rand    :67ed56bfcf99825f0c6b349369da30ab
PIN     :7885b515e84b1f082cc499976f1725ce
round  1:67ed56bfcf99825f0c6b349369da30bb
Key [ 1]:7885b515e84b1f082cc499976f1725ce
Key [ 2]:72445901fdaf506beb036f4412512248
round  2:6b160b66a1f6c26c1f3432f463ef5aa1
Key [ 3]:59f0e4982e97633e5e7fd133af8f2c5b
Key [ 4]:b4946ec77a41bf7c729d191e33d458ab
round  3:3f22046c964c3e5ca2a26ec9a76a9f67
added ->:580f5ad359e5c003ae0da25ace44cfdc
Key [ 5]:eb0b839f97bdf534183210678520bbef
Key [ 6]:cff0bc4a94e5c8b2a2d24d9f59031e19
round  4:87aa61fc0ff88e744c195249b9a33632
Key [ 7]:592430f14d8f93db95dd691af045776d
Key [ 8]:3b55b404222bf445a6a2ef5865247695
round  5:83dcf592a854226c4dcd94e1ecf1bc75
Key [ 9]:a9714b86319ef343a28b87456416bd52
Key [10]:e6598b24390b3a0bf2982747993b0d78
round  6:dee0d13a52e96bcf7c72045a21609fc6
Key [11]:62051d8c51973073bff959b032c6e1e2
Key [12]:29e94f4ab73296c453c833e217a1a85b
round  7:08488005761e6c7c4dbb203ae453fe3a
Key [13]:0e255970b3e2fc235f59fc5acb10e8ce
Key [14]:d0dfbb3361fee6d4ffe45babf1cd7abf
round  8:0d81e89bddde7a7065316c47574feb8f
Key [15]:c12eee4eb38b7a171f0f736003774b40
Key [16]:8f962523f1c0abd9a087a0dfb11643d3
Key [17]:24be1c66cf8b022f12f1fb4c60c93fd1
Ka      :04435771e03a9daceb8bb1a493ee9bd8
---------------------------------------
rand    :40a94509238664f244ff8e3d13b119d3
PIN     :1ce44839badde30396d03c4c36f23006
round  1:40a94509238664f244ff8e3d13b119c3
Key [ 1]:1ce44839badde30396d03c4c36f23006
Key [ 2]:6dd97a8f91d628be4b18157af1a9dcba
round  2:0eac5288057d9947a24eabc1744c4582
Key [ 3]:fef9583d5f55fd4107ad832a725db744
Key [ 4]:fc3893507016d7c1db2bd034a230a069
round  3:60b424f1082b0cc3bd61be7b4c0155f0
added ->:205d69f82bb17031f9604c465fb26e33
Key [ 5]:0834d04f3e7e1f7f85f0c1db685ab118
Key [ 6]:1852397f9a3723169058e9b62bb3682b
round  4:2c6b65a49d66af6566675afdd6fa7d7d
Key [ 7]:6c10da21d762ae4ac1ba22a96d9007b4
Key [ 8]:9aa23658b90470a78d686344b8a9b0e7
round  5:a2c537899665113a42f1ac24773bdc31
Key [ 9]:137dee3bf879fe7bd02fe6d888e84f16
Key [10]:466e315a1863f47d0f93bc6827cf3450
round  6:e26982980d79b21ed3e20f8c3e71ba96
Key [11]:0b33cf831465bb5c979e6224d7f79f7c
Key [12]:92770660268ede827810d707a0977d73
```

```
round  7:e7b063c4e2e3110b89b7e1631c762dd5
Key [13]:7be30ae4961cf24ca17625a77bb7a9f8
Key [14]:be65574a33ae30e6e82dbd2826d3cc1a
round  8:7a963e37b2c2e76b489cfe40a2cf00e5
Key [15]:ed0ba7dd30d60a5e69225f0a33011e5b
Key [16]:765c990f4445e52b39e6ed6105ad1c4f
Key [17]:52627bf9f35d94f30d5b07ef15901adc
Ka      :9cde4b60f9b5861ed9df80858bac6f7f


========================================
```

## 10.4 TESTS OF E22 WITH PIN AUGMENTING

for PIN lengths 1,...,16 bytes

```
rand    :24b101fd56117d42c0545a4247357048
PIN length =16 octets
PIN     :fd397c7f5c1f937cdf82d8816cc377e2
round  1:24b101fd56117d42c0545a4247357058
Key [ 1]:fd397c7f5c1f937cdf82d8816cc377e2
Key [ 2]:0f7aac9c9b53f308d9fdbf2c78e3c30e
round  2:838edfe1226266953ccba8379d873107
Key [ 3]:0b8ac18d4bb44fad2efa115e43945abc
Key [ 4]:887b16b062a83bfa469772c25b456312
round  3:8cd0c9283120aba89a7f9d635dd4fe3f
added ->:a881cad5673128ea5ad3f7211a096e67
Key [ 5]:2248cbe6d299e9d3e8fd35a91178f65b
Key [ 6]:b92af6237385bd31f8fb57fb1bdd824e
round  4:2648d9c618a622b10ef80c4dbaf68b99
Key [ 7]:2bf5ffe84a37878ede2d4c30be60203b
Key [ 8]:c9cb6cec60cb8a8f29b99fcf3e71e40f
round  5:b5a7d9e96f68b14ccebf361de3914d0f
Key [ 9]:5c2f8a702e4a45575b103b0cce8a91c6
Key [10]:d453db0c9f9ddbd11e355d9a34d9b11b
round  6:632a091e7eefe1336857ddafd1ff3265
Key [11]:32805db7e59c5ed4acabf38d27e3fece
Key [12]:fde3a8eedfa3a12be09c1a8a00890fd7
round  7:048531e9fd3efa95910540150f8b137b
Key [13]:def07eb23f3a378f059039a2124bc4c2
Key [14]:2608c58f23d84a09b9ce95e5caac1ab4
round  8:461814ec7439d412d0732f0a6f799a6a
Key [15]:0a7ed16481a623e56ee1442ffa74f334
Key [16]:12add59aca0d19532f1516979954e369
Key [17]:dd43d02d39ffd6a386a4b98b4ac6eb23
Ka      :a5f2adf328e4e6a2b42f19c8b74ba884
----------------------------------------
rand    :321964061ac49a436f9fb9824ac63f8b
PIN length =15 octets
PIN     :ad955d58b6b8857820ac1262d617a6
address :0314c0642543
round  1:321964061ac49a436f9fb9824ac63f9b
```

*Sample Data*

```
Key [ 1]:ad955d58b6b8857820ac1262d617a603
Key [ 2]:f281736f68e3d30b2ac7c67f125dc416
round  2:7c4a4ece1398681f4bafd309328b7770
Key [ 3]:43c157f4c8b360387c32ab330f9c9aa8
Key [ 4]:3a3049945a298f6d076c19219c47c3cb
round  3:9672b00738bdfaf9bd92a855bc6f3afb
added ->:a48b1401228194bad23161d7f6357960
Key [ 5]:c8e2eaa6d73b7de18f3228ab2173bc69
Key [ 6]:8623f44488222e66a293677cf30bf2bb
round  4:9b30247aad3bf133712d034b46d21c68
Key [ 7]:f3e500902fba31db9bae50ef30e484a4
Key [ 8]:49d4b1137c18f4752dd9955a5a8d2f43
round  5:4492c25fda08083a768b4b5588966b23
Key [ 9]:9d59c451989e74785cc097eda7e42ab8
Key [10]:251de25f3917dcd99c18646107a641fb
round  6:21ae346635714d2623041f269978c0ee
Key [11]:80b8f78cb1a49ec0c3e32a238e60fddf
Key [12]:beb84f4d20a501e4a24ecfbde481902b
round  7:9b56a3d0f8932f20c6a77a229514fb00
Key [13]:852571b44f35fd9d9336d3c1d2506656
Key [14]:d0a0d510fb06ba76e69b8ee3ebc1b725
round  8:6cd8492b2fd31a86978bcdf644eb08a8
Key [15]:c7ffd523f32a874ed4a93430a25976de
Key [16]:16cdcb25e62964876d951fdcc07030d3
Key [17]:def32c0e12596f9582e5e3c52b303f52
Ka      :c0ec1a5694e2b48d54297911e6c98b8f
----------------------------------------
rand    :d4ae20c80094547d7051931b5cc2a8d6
PIN length =14 octets
PIN     :e1232e2c5f3b833b3309088a87b6
address :fabecc58e609
round  1:d4ae20c80094547d7051931b5cc2a8c6
Key [ 1]:e1232e2c5f3b833b3309088a87b6fabe
Key [ 2]:5f0812b47cd3e9a30d7707050fffa1f2
round  2:1f45f16be89794bef33e4547c9c0916a
Key [ 3]:77b681944763244ffa3cd71b248b79b5
Key [ 4]:e2814e90e04f485958ce58c9133e2be6
round  3:b10d2f4ac941035263cee3552d774d2f
added ->:65bb4f82c9d5572f131f764e7139f5e9
Key [ 5]:520acad20801dc639a2c6d66d9b79576
Key [ 6]:c72255cdb61d42be72bd45390dd25ba5
round  4:ead4dc34207b6ea721c62166e155aaad
Key [ 7]:ebf04c02075bf459ec9c3ec06627d347
Key [ 8]:a1363dd2812ee800a4491c0c74074493
round  5:f507944f3018e20586d81d7f326aae9d
Key [ 9]:b0b6ba79493dc833d7f425be7b8dadb6
Key [10]:08cd23e536b9b9b53e85eb004cba3111
round  6:fff450f4302a2b3571e8405e148346da
Key [11]:fec22374c6937dcd26171f4d2edfada3
Key [12]:0f1a8ef5979c69ff44f620c2e007b6e4
round  7:de558779589897f3402a90ee78c3f921
Key [13]:901fb66f0779d6aad0c0fba1fe812cb5
```

*Sample Data*

```
Key [14]:a0cab3cd15cd23603adc8d4474efb239
round  8:b2df0aa0c9f07fbbaa02f510a29cf540
Key [15]:18edc3f4296dd6f1dea13f7c143117a1
Key [16]:8d3d52d700a379d72ded81687f7546c7
Key [17]:5927badfe602f29345f840bb53e1dea6
Ka      :d7b39be13e3692c65b4a9e17a9c55e17
----------------------------------------
rand    :272b73a2e40db52a6a61c6520549794a
PIN length =13 octets
PIN     :549f2694f353f5145772d8ae1e
address :20487681eb9f
round  1:272b73a2e40db52a6a61c6520549795a
Key [ 1]:549f2694f353f5145772d8ae1e204876
Key [ 2]:42c855593d66b0c458fd28b95b6a5fbf
round  2:d7276dc8073f7677c31f855bde9501e2
Key [ 3]:75d0a69ae49a2da92e457d767879df52
Key [ 4]:b3aa7e7492971afaa0fb2b64827110df
round  3:71aae503831133d19bc452da4d0e409b
added ->:56d558a1671ee8fbf12518884857b9c1
Key [ 5]:9c8cf1604a98e9a503c342e272de5cf6
Key [ 6]:d35bc2df6b85540a27642106471057d9
round  4:f41a709c89ea80481aa3d2b9b2a9f8ca
Key [ 7]:b454dda74aeb4eff227ba48a58077599
Key [ 8]:bcba6aec050116aa9b7c6a9b7314d796
round  5:20fdda20f4a26b1bd38eb7f355a7be87
Key [ 9]:d41f8a9de0a716eb7167a1b6e321c528
Key [10]:53534499822477 82d168ab43f17bc4d8
round  6:a70e316997eeed49a5a9ef9ba5e913b5
Key [11]:32cbc9cf1a81e36a45153972347ce4ac
Key [12]:5747619006cf4ef834c749f2c4b9feb6
round  7:e66f2317a825f589f76b47b6aa6e73fb
Key [13]:f9b68beba0a09d2a570a7dc88cc3c3c2
Key [14]:55718f9a4f0b1f9484e8c6b186a41a4b
round  8:5f68f940440a9798e074776019804ada
Key [15]:4ecc29be1b4d78433f6aa30db974a7fb
Key [16]:8470a066ffb00cda7b08059599f919f5
Key [17]:f39a36d74e960a051e1ca98b777848f4
Ka      :9ac64309a37c25c3b4a584fc002a1618
----------------------------------------
rand    :7edb65f01a2f45a2bc9b24fb3390667e
PIN length =12 octets
PIN     :2e5a42797958557b23447ca8
address :04f0d2737f02
round  1:7edb65f01a2f45a2bc9b24fb3390666e
Key [ 1]:2e5a42797958557b23447ca804f0d273
Key [ 2]:18a97c856561eb23e71af8e9e1be4799
round  2:3436e12db8ffdc1265cb5a86da2fed0b
Key [ 3]:7c0908dcbc73201e17c4f7aa1ab8aec8
Key [ 4]:7cb58833602fbe4194c7cc797ce8c454
round  3:caed6af4226f67e4ad1914620803ef2a
added ->:b4c8cf04389eac4611b438993b935544
Key [ 5]:f4dce7d607b5234562d0ebb2267b08b8
```

*Sample Data*

```
Key [ 6]:560b75c5545751fd8fa99fa4346e654b
round  4:ee67c87d6f74bb75db98f68bff0192c1
Key [ 7]:32f10cefd8d3e6424c6f91f1437808af
Key [ 8]:a934a46045be30fb3be3a5f3f7b18837
round  5:792398dcbeb8d10bdb07ae3c819e943c
Key [ 9]:a0f12e97c677a0e8ac415cd2c8a7ca88
Key [10]:e27014c908785f5ca03e8c6a1da3bf13
round  6:e778b6e0c3e8e7edf90861c7916d97a8
Key [11]:1b4a4303bcc0b2e0f41c72d47654bd9f
Key [12]:4b1302a50046026d6c9054fc8387965a
round  7:1fafddc7efa5f04c1dec1869d3f2d9bb
Key [13]:58c334bb543d49eca562cdbe0280e0fc
Key [14]:bdb60d383c692d06476b76646c8dec48
round  8:3d7c326d074bd6aa222ea050f04a3c7f
Key [15]:78c0162506be0b5953e8403c01028f93
Key [16]:24d7dbbe834dbd7b67f57fcf0d39d60f
Key [17]:2e74f1f3331c0f6585e87b2f715e187e
Ka      :d3af4c81e3f482f062999dee7882a73b
----------------------------------------
rand    :26a92358294dce97b1d79ec32a67e81a
PIN length =11 octets
PIN     :05fbad03f52fa9324f7732
address :b9ac071f9d70
round  1:26a92358294dce97b1d79ec32a67e80a
Key [ 1]:05fbad03f52fa9324f7732b9ac071f9d
Key [ 2]:2504c9691c04a18480c8802e922098c0
round  2:0be20e3d76888e57b6bf77f97a8714fb
Key [ 3]:576b2791d1212bea8408212f2d43e77e
Key [ 4]:90ae36dcce8724adb618f912d1b27297
round  3:1969667060764453257d906b7e58bd5b
added ->:3f12892849c312c494542ea854bfa551
Key [ 5]:bc492c42c9e87f56ec31af5474e9226e
Key [ 6]:c135d1dbed32d9519acfb4169f3e1a10
round  4:ac404205118fe771e54aa6f392da1153
Key [ 7]:83ccbdbbaf17889b7d18254dc9252fa1
Key [ 8]:80b90a1767d3f2848080802764e21711
round  5:41795e89ae9a0cf776ffece76f47fd7a
Key [ 9]:cc24e4a86e8eed129118fd3d5223a1dc
Key [10]:7b1e9c0eb9dab083574be7b7015a62c9
round  6:29ca9e2f87ca00370ef1633505bfba4b
Key [11]:888e6d88cf4beb965cf7d4f32b696baa
Key [12]:6d642f3e5510b0b043a44daa2cf5eec0
round  7:81fc891c3c6fd99acc00028a387e2366
Key [13]:e224f85da2ab63a23e2a3a036e421358
Key [14]:c8dc22aaa739e2cb85d6a0c08226c7d0
round  8:e30b537e7a000e3d2424a9c0f04c4042
Key [15]:a969aa818c6b324bae391bedcdd9d335
Key [16]:6974b6f2f07e4c55f2cc0435c45bebd1
Key [17]:134b925ebd98e6b93c14aee582062fcb
Ka      :be87b44d079d45a08a71d15208c5cb50
----------------------------------------
rand    :0edef05327eab5262430f21fc91ce682
```

*Sample Data*

```
PIN length =10 octets
PIN     :8210e47390f3f48c32b3
address :7a3cdfe377d1
round  1:0edef05327eab5262430f21fc91ce692
Key [ 1]:8210e47390f3f48c32b37a3cdfe377d1
Key [ 2]:c6be4c3e425e749b620a94c779e33a7e
round  2:07ca3c7a7a6bcbc31d79a856d9cffc0e
Key [ 3]:2587cec2a4b8e4f996a9ed664350d5dd
Key [ 4]:70e4bf72834d9d3dbb7eb2c239216dc0
round  3:792ad2ac4e4559d1463714d2f161b6f4
added ->:7708c2ff692f0ef7626706cd387d9c66
Key [ 5]:6696e1e7f8ac037e1fff3598f0c164e2
Key [ 6]:23dbfe4d0b561bea08fbcef25e49b648
round  4:7d8c71a9d7fbdcbd851bdf074550b100
Key [ 7]:b03648acd021550edee904431a02f00c
Key [ 8]:cb169220b7398e8f077730aa4bf06baa
round  5:b6fcaa45064ffd557e4b7b30cfbb83e0
Key [ 9]:af602c2ba16a454649951274c2be6527
Key [10]:5d60b0a7a09d524143eca13ad680bc9c
round  6:b3416d391a0c26c558843debd0601e9e
Key [11]:9a2f39bfe558d9f562c5f09a5c3c0263
Key [12]:72cae8eebd7fabd9b1848333c2aab439
round  7:abe4b498d9c36ea97b8fd27d7f813913
Key [13]:15f27ea11e83a51645d487b81371d7dc
Key [14]:36083c8666447e03d33846edf444eb12
round  8:8032104338a945ba044d102eabda3b22
Key [15]:0a3a8977dd48f3b6c1668578befadd02
Key [16]:f06b6675d78ca0ee5b1761bdcdab516d
Key [17]:cbc8a7952d33aa0496f7ea2d05390b23
Ka      :bf0706d76ec3b11cce724b311bf71ff5
----------------------------------------
rand    :86290e2892f278ff6c3fb917b020576a
PIN length = 9 octets
PIN     :3dcdffcfd086802107
address :791a6a2c5cc3
round  1:86290e2892f278ff6c3fb917b0205765
Key [ 1]:3dcdffcfd086802107791a6a2c5cc33d
Key [ 2]:b4962f40d7bb19429007062a3c469521
round  2:1ec59ffd3065f19991872a7863b0ef02
Key [ 3]:eb9ede6787dd196b7e340185562bf28c
Key [ 4]:2964e58aacf7287d1717a35b100ae23b
round  3:f817406f1423fc2fe33e25152679eaaf
added ->:7e404e47861574d08f7dde02969941ca
Key [ 5]:6abf9a314508fd61e486fa4e376c3f93
Key [ 6]:6da148b7ee2632114521842cbb274376
round  4:e9c2a8fac22b8c7cf0c619e2b3f890ed
Key [ 7]:df889cc34fda86f01096d52d116e620d
Key [ 8]:5eb04b147dc39d1974058761ae7b73fc
round  5:444a8aac0efee1c02f8d38f8274b7b28
Key [ 9]:8426cc59eee391b2bd50cf8f1efef8b3
Key [10]:8b5d220a6300ade418da791dd8151941
round  6:9185f983db150b1bccab1e5c12eb63a1
```

*Sample Data*

```
Key [11]:82ba4ddef833f6a4d18b07aa011f2798
Key [12]:ce63d98794682054e73d0359dad35ec4
round  7:5eded2668f5916dfd036c09e87902886
Key [13]:da794357652e80c70ad8b0715dbe33d6
Key [14]:732ef2c0c3220b31f3820c375e27bb29
round  8:88a5291b4acbba009a85b7dd6a834b3b
Key [15]:3ce75a61d4b465b70c95d7ccd5799633
Key [16]:5df9bd2c3a17a840cdaafb76c171db7c
Key [17]:3f8364b089733d902bccb0cd3386846f
Ka      :cdb0cc68f6f6fbd70b46652de3ef3ffb
----------------------------------------
rand    :3ab52a65bb3b24a08eb6cd284b4b9d4b
PIN length = 8 octets
PIN     :d0fb9b6838d464d8
address :25a868db91ab
round  1:3ab52a65bb3b24a08eb6cd284b4b9d45
Key [ 1]:d0fb9b6838d464d825a868db91abd0fb
Key [ 2]:2573f47b49dad6330a7a9155b7ae8ba1
round  2:ad2ffdff408fcfab44941016a9199251
Key [ 3]:d2c5b8fb80cba13712905a589adaee71
Key [ 4]:5a3381511b338719fae242758dea0997
round  3:2ddc17e570d7931a2b1d13f6ace928a5
added ->:17914180cb12b7baa5d3e0dee734c5e0
Key [ 5]:e0a4d8ac27fbe2783b7bcb3a36a6224d
Key [ 6]:949324c6864deac3eca8e324853e11c3
round  4:62c1db5cf31590d331ec40ad692e8df5
Key [ 7]:6e67148088a01c2d4491957cc9ddc4aa
Key [ 8]:557431deab7087bb4c03fa27228f60c6
round  5:9c8933bc361f4bde4d1bda2b5f8bb235
Key [ 9]:a2551aca53329e70ade3fd2bb7664697
Key [10]:05d0ad35de68a364b54b56e2138738fe
round  6:9156db34136aa06655bf28a05be0596a
Key [11]:1616a6b13ce2f2895c722e8495181520
Key [12]:b12e78a1114847b01f6ed2f5a1429a23
round  7:84dcc292ed836c1c2d523f2a899a2ad5
Key [13]:316e144364686381944e95afd8a026bb
Key [14]:1ab551b88d39d97ea7a9fe136dbfe2e1
round  8:87bdcac878d777877f4eccf042cfee5e
Key [15]:70e21ab08c23c7544524b64492b25cc9
Key [16]:35f730f2ae2b950a49a1bf5c8b9f8866
Key [17]:2f16924c22db8b74e2eadf1ba4ebd37c
Ka      :983218718ca9aa97892e312d86dd9516
----------------------------------------
rand    :a6dc447ff08d4b366ff96e6cf207e179
PIN length = 7 octets
PIN     :9c57e10b4766cc
address :54ebd9328cb6
round  1:a6dc447ff08d4b366ff96e6cf207e174
Key [ 1]:9c57e10b4766cc54ebd9328cb69c57e1
Key [ 2]:00a609f4d61db26993c8177e3ee2bba8
round  2:1ed26b96a306d7014f4e5c9ee523b73d
Key [ 3]:646d7b5f9aaa528384bda3953b542764
```

*Sample Data*

```
Key [ 4]:a051a42212c0e9ad5c2c248259aca14e
round  3:a53f526db18e3d7d53edbfc9711041ed
added ->:031b9612411b884b3ce62da583172299
Key [ 5]:d1bd5e64930e7f838d8a33994462d8b2
Key [ 6]:5dc7e2291e32435665ebd6956bec3414
round  4:9438be308ec83f35c560e2796f4e0559
Key [ 7]:10552f45af63b0f15e2919ab37f64fe7
Key [ 8]:c44d5717c114a58b09207392ebe341f8
round  5:b79a7b14386066d339f799c40479cb3d
Key [ 9]:6886e47b782325568eaf59715a75d8ff
Key [10]:8e1e335e659cd36b132689f78c147bda
round  6:ef232462228aa166438d10c34e17424b
Key [11]:8843efeedd5c2b7c3304d647f932f4d1
Key [12]:13785aaedd0adf67abb4f01872392785
round  7:02d133fe40d15f1073673b36bba35abd
Key [13]:837d7ca2722419e6be3fae35900c3958
Key [14]:93f8442973e7fccf2e7232d1d057c73a
round  8:275506a3d08c84e94cc58ed60054505e
Key [15]:8a7a9edffa3c52918bc6a45f57d91f5d
Key [16]:f214a95d777f763c56109882c4b52c84
Key [17]:10e2ee92c5ea1ddc5eb010e55510c403
Ka      :9cd6650ead86323e87cafb1ff516d1e0
----------------------------------------
rand    :3348470a7ea6cc6eb81b40472133262c
PIN length = 6 octets
PIN     :fcad169d7295
address :430d572f8842
round  1:3348470a7ea6cc6eb81b404721332620
Key [ 1]:fcad169d7295430d572f8842fcad169d
Key [ 2]:b3479d4d4fd178c43e7bc5b0c7d8983c
round  2:af976da9225066d563e10ab955e6fc32
Key [ 3]:7112462b37d82dd81a2a35d9eb43cb7c
Key [ 4]:c5a7030f8497945ac7b84600d1d161fb
round  3:d08f826ebd55a0bd7591c19a89ed9bde
added ->:e3d7c964c3fb6cd3cdac01dda820c1fe
Key [ 5]:84b0c6ef4a63e4dff19b1f546d683df5
Key [ 6]:f4023edfc95d1e79ed4bb4de9b174f5d
round  4:6cd952785630dfc7cf81eea625e42c5c
Key [ 7]:ea38dd9a093ac9355918632c90c79993
Key [ 8]:dbba01e278ddc76380727f5d7135a7de
round  5:93573b2971515495978264b88f330f7f
Key [ 9]:d4dc3a31be34e412210fafa6eca00776
Key [10]:39d1e190ee92b0ff16d92a8be58d2fa0
round  6:b3f01d5e7fe1ce6da7b46d8c389baf47
Key [11]:1eb081328d4bcf94c9117b12c5cf22ac
Key [12]:7e047c2c552f9f1414d946775fabfe30
round  7:0b833bff6106d5bae033b4ce5af5a924
Key [13]:e78e685d9b2a7e29e7f2a19d1bc38ebd
Key [14]:1b582272a3121718c4096d2d8602f215
round  8:23de0bbdc70850a7803f4f10c63b2c0f
Key [15]:8569e860530d9c3d48a0870dac33f676
Key [16]:6966b528fdd1dc222527052c8f6cf5a6
```

*Sample Data*

```
Key [17]:a34244c757154c53171c663b0b56d5c2
Ka      :98f1543ab4d87bd5ef5296fb5e3d3a21
----------------------------------------
rand    :0f5bb150b4371ae4e5785293d22b7b0c
PIN length = 5 octets
PIN     :b10d068bca
address :b44775199f29
round  1:0f5bb150b4371ae4e5785293d22b7b07
Key [ 1]:b10d068bcab44775199f29b10d068bca
Key [ 2]:aec70d1048f1bbd2c18040318a8402ad
round  2:342d2b79d7fb7cd110379742b9842c79
Key [ 3]:6d8d5cf338f29ef4420639ef488e4fa9
Key [ 4]:a1584117541b759ba6d9f7eb2bedcbba
round  3:9407e8e3e810603921bf81cfda62770a
added ->:9b6299b35c477addc437d35c088df20d
Key [ 5]:09a20676666aeed6f22176274eb433f4
Key [ 6]:840472c001add5811a054be5f5c74754
round  4:9a3ba953225a7862c0a842ed3d0b2679
Key [ 7]:fad9e45c8bf70a972fcd9bff0e8751f5
Key [ 8]:e8f30ff666dfd212263416496ff3b2c2
round  5:2c573b6480852e875df34b28a5c44509
Key [ 9]:964cdba0cf8d593f2fc40f96daf8267a
Key [10]:bcd65c11b13e1a70bcd4aafba8864fe3
round  6:21b0cc49e880c5811d24dee0194e6e9e
Key [11]:468c8548ea9653c1a10df6288dd03c1d
Key [12]:5d252d17af4b09d3f4b5f7b5677b8211
round  7:e6d6bdcd63e1d37d9883543ba86392fd
Key [13]:e814bf307c767428c67793dda2df95c7
Key [14]:4812b979fdc20f0ff0996f61673a42cc
round  8:e3dde7ce6bd7d8a34599aa04d6a760ab
Key [15]:5b1e2033d1cd549fc4b028146eb5b3b7
Key [16]:0f284c14fb8fe706a5343e3aa35af7b1
Key [17]:b1f7a4b7456d6b577fded6dc7a672e37
Ka      :c55070b72bc982adb972ed05d1a74ddb
----------------------------------------
rand    :148662a4baa73cfadb55489159e476e1
PIN length = 4 octets
PIN     :fb20f177
address :a683bd0b1896
round  1:148662a4baa73cfadb55489159e476eb
Key [ 1]:fb20f177a683bd0b1896fb20f177a683
Key [ 2]:47266cefbfa468ca7916b458155dc825
round  2:3a942eb6271c3f4e433838a5d3fcbd27
Key [ 3]:688853a6d6575eb2f6a2724b0fbc133b
Key [ 4]:7810df048019634083a2d9219d0b5fe0
round  3:9c835b98a063701c0887943596780769
added ->:8809bd3c1a0aace6d3dcdca4cf5c7d82
Key [ 5]:c78f6dcf56da1bbd413828b33f5865b3
Key [ 6]:eb3f3d407d160df3d293a76d1a513c4a
round  4:7e68c4bafa020a4a59b5a1968105bab5
Key [ 7]:d330e038d6b19d5c9bb0d7285a360064
Key [ 8]:9bd3ee50347c00753d165faced702d9c
```

*Sample Data*

```
round  5:227bad0cf0838bdb15b3b3872c24f592
Key [ 9]:9543ad0fb3fe74f83e0e2281c6d4f5f0
Key [10]:746cd0383c17e0e80e6d095a87fd0290
round  6:e026e98c71121a0cb739ef6f59e14d26
Key [11]:fa28bea4b1c417536608f11f406ea1dd
Key [12]:3aee0f4d21699df9cb8caf5354a780ff
round  7:cd6a6d8137d55140046f8991da1fa40a
Key [13]:372b71bc6d1aa6e785358044fbcf05f4
Key [14]:00a01501224c0405de00aa2ce7b6ab04
round  8:52cd7257fe8d0c782c259bcb6c9f5942
Key [15]:c7015c5c1d7c030e00897f104a006d4a
Key [16]:260a9577790c62e074e71e19fd2894df
Key [17]:c041b7a231493acd15ddcdaee94b9f52
Ka      :7ec864df2f1637c7e81f2319ae8f4671
----------------------------------------
rand    :193a1b84376c88882c8d3b4ee93ba8d5
PIN length = 3 octets
PIN     :a123b9
address :4459a44610f6
round  1:193a1b84376c88882c8d3b4ee93ba8dc
Key [ 1]:a123b94459a44610f6a123b94459a446
Key [ 2]:5f64d384c8e990c1d25080eb244dde9b
round  2:3badbd58f100831d781ddd3ccedefd3f
Key [ 3]:5abc00eff8991575c00807c48f6dbea5
Key [ 4]:127521158ad6798fb6479d1d2268abe6
round  3:0b53075a49c6bf2df2421c655fdedf68
added ->:128d22de7e3247a5decf572bb61987b4
Key [ 5]:f2a1f620448b8e56665608df2ab3952f
Key [ 6]:7c84c0af02aad91dc39209c4edd220b1
round  4:793f4484fb592e7a78756fd4662f990d
Key [ 7]:f6445b647317e7e493bb92bf6655342f
Key [ 8]:3cae503567c63d3595eb140ce60a84c0
round  5:9e46a8df925916a342f299a8306220a0
Key [ 9]:734ed5a806e072bbecb4254993871679
Key [10]:cda69ccb4b07f65e3c8547c11c0647b8
round  6:6bf9cd82c9e1be13fc58eae0b936c75a
Key [11]:c48e531d3175c2bd26fa25cc8990e394
Key [12]:6d93d349a6c6e9ff5b26149565b13d15
round  7:e96a9871471240f198811d4b8311e9a6
Key [13]:5c4951e85875d663526092cd4cbdb667
Key [14]:f19f7758f5cde86c3791efaf563b3fd0
round  8:e94ca67d3721d5fb08ec069191801a46
Key [15]:bf0c17f3299b37d984ac938b769dd394
Key [16]:7edf4ad772a6b9048588f97be25bde1c
Key [17]:6ee7ba6afefc5b561abbd8d6829e8150
Ka      :ac0daabf17732f632e34ef193658bf5d
----------------------------------------
rand    :1453db4d057654e8eb62d7d62ec3608c
PIN length = 2 octets
PIN     :3eaf
address :411fbbb51d1e
round  1:1453db4d057654e8eb62d7d62ec36084
```

*Sample Data*

```
Key [ 1]:3eaf411fbbb51d1e3eaf411fbbb51d1e
Key [ 2]:c3a1a997509f00fb4241aba607109c64
round  2:0b78276c1ebc65707d38c9c5fa1372bd
Key [ 3]:3c729833ae1ce7f84861e4dbad6305cc
Key [ 4]:c83a43c3a66595cb8136560ed29be4ff
round  3:23f3f0f6441563d4c202cee0e5cb2335
added ->:3746cbbb418bb73c2964a536cb8e83b1
Key [ 5]:18b26300b86b70acdd1c8f5cbc7c5da8
Key [ 6]:04efc75309b98cd8f1cef5513c18e41e
round  4:c61afa90d3c14bdf588320e857afdc00
Key [ 7]:517c789cecadc455751af73198749fb8
Key [ 8]:fd9711f913b5c844900fa79dd765d0e2
round  5:a8a0e02ceb556af8bfa321789801183a
Key [ 9]:bb5cf30e7d3ceb930651b1d16ee92750
Key [10]:3d97c7862ecab42720e984972f8efd28
round  6:0b58e922438d224db34b68fca9a5ea12
Key [11]:4ce730344f6b09e449dcdb64cd466666
Key [12]:38828c3a56f922186adcd9b713cdcc31
round  7:b90664c4ac29a8b4bb26debec9ffc5f2
Key [13]:d30fd865ea3e9edcff86a33a2c319649
Key [14]:1fdb63e54413acd968195ab6fa424e83
round  8:6934de3067817cefd811abc5736c163b
Key [15]:a16b7c655bbaa262c807cba8ae166971
Key [16]:7903dd68630105266049e23ca607cda7
Key [17]:888446f2d95e6c2d2803e6f4e815ddc9
Ka      :1674f9dc2063cc2b83d3ef8ba692ebef
----------------------------------------
rand    :1313f7115a9db842fcedc4b10088b48d
PIN length = 1 octets
PIN     :6d
address :008aa9be62d5
round  1:1313f7115a9db842fcedc4b10088b48a
Key [ 1]:6d008aa9be62d56d008aa9be62d56d00
Key [ 2]:46ebfeafb6657b0a1984a8dc0893accf
round  2:839b23b83b5701ab095bafd162ec0ac7
Key [ 3]:8e15595edcf058af62498ee3c1dc6098
Key [ 4]:dd409c3444e94b9cc08396ae967542a0
round  3:c0a2010cc44f2139427f093f4f97ae68
added ->:d3b5f81d9eecd97bbe6ccd8e4f1f62e2
Key [ 5]:487deff5d519f6a6481e947b926f633c
Key [ 6]:5b4b6e3477ed5c2c01f6e607d3418963
round  4:1a5517a0efad3575931d8ea3bee8bd07
Key [ 7]:34b980088d2b5fd6b6a2aceeda99c9c4
Key [ 8]:e7d06d06078acc4ecdbc8da800b73078
round  5:d3ce1fdfe716d72c1075ff37a8a2093f
Key [ 9]:7d375bad245c3b757380021af8ecd408
Key [10]:14dac4bc2f4dc4929a6cceec47f4c3a3
round  6:47e90cb55be6e8dd0f583623c2f2257b
Key [11]:66cfda3c63e464b05e2e7e25f8743ad7
Key [12]:77cfccda1ad380b9fdf1df10846b50e7
round  7:f866ae6624f7abd4a4f5bd24b04b6d43
Key [13]:3e11dd84c031a470a8b66ec6214e44cf
```

*Sample Data*

```
Key [14]:2f03549bdb3c511eea70b65ddbb08253
round  8:02e8e17cf8be4837c9c40706b613dfa8
Key [15]:e2f331229ddfcc6e7bea08b01ab7e70c
Key [16]:b6b0c3738c5365bc77331b98b3fba2ab
Key [17]:f5b3973b636119e577c5c15c87bcfd19
Ka      :38ec0258134ec3f08461ae5c328968a1

========================================
```

## 10.5 FOUR TESTS OF E3

```
rand    :00000000000000000000000000000000
aco     :48afcdd4bd40fef76693b113
key     :00000000000000000000000000000000
round  1:00000000000000000000000000000000
Key [ 1]:00000000000000000000000000000000
Key [ 2]:4697b1baa3b7100ac537b3c95a28ac64
round  2:78d19f9307d2476a523ec7a8a026042a
Key [ 3]:ecabaac66795580df89af66e66dc053d
Key [ 4]:8ac3d8896ae9364943bfebd4969b68a0
round  3:600265247668dda0e81c07bbb30ed503
Key [ 5]:5d57921fd5715cbb22c1be7bbc996394
Key [ 6]:2a61b8343219fdfb1740e6511d41448f
round  4:d7552ef7cc9dbde568d80c2215bc4277
Key [ 7]:dd0480dee731d67f01a2f739da6f23ca
Key [ 8]:3ad01cd1303e12a1cd0fe0a8af82592c
round  5:fb06bef32b52ab8f2a4f2b6ef7f6d0cd
Key [ 9]:7dadb2efc287ce75061302904f2e7233
Key [10]:c08dcfa981e2c4272f6c7a9f52e11538
round  6:b46b711ebb3cf69e847a75f0ab884bdd
Key [11]:fc2042c708e409555e8c147660ffdfd7
Key [12]:fa0b21001af9a6b9e89e624cd99150d2
round  7:c585f308ff19404294f06b292e978994
Key [13]:18b40784ea5ba4c80ecb48694b4e9c35
Key [14]:454d54e5253c0c4a8b3fcca7db6baef4
round  8:2665fadb13acf952bf74b4ab12264b9f
Key [15]:2df37c6d9db52674f29353b0f011ed83
Key [16]:b60316733b1e8e70bd861b477e2456f1
Key [17]:884697b1baa3b7100ac537b3c95a28ac
round  1:5d3ecb17f26083df0b7f2b9b29aef87c
Key [ 1]:e9e5dfc1b3a79583e9e5dfc1b3a79583
Key [ 2]:7595bf57e0632c59f435c16697d4c864
round  2:de6fe85c5827233fe22514a16f321bd8
Key [ 3]:e31b96afcc75d286ef0ae257cbbc05b7
Key [ 4]:0d2a27b471bc0108c6263aff9d9b3b6b
round  3:7cd335b50d09d139ea6702623af85edb
added ->:211100a2ff6954e6e1e62df913a656a7
Key [ 5]:98d1eb5773cf59d75d3b17b3bc37c191
Key [ 6]:fd2b79282408ddd4ea0aa7511133336f
round  4:991dccb3201b5b1c4ceff65a3711e1e9
Key [ 7]:331227756638a41d57b0f7e071ee2a98
```

*Sample Data*

```
Key [ 8]:aa0dd8cc68b406533d0f1d64aabacf20
round  5:18768c7964818805fe4c6ecae8a38599
Key [ 9]:669291b0752e63f806fce76f10e119c8
Key [10]:ef8bdd46be8ee0277e9b78adef1ec154
round  6:82f9aa127a72632af43d1a17e7bd3a09
Key [11]:f3902eb06dc409cfd78384624964bf51
Key [12]:7d72702b21f97984a721c99b0498239d
round  7:1543d7870bf2d6d6efab3cbf62dca97d
Key [13]:532e60bceaf902c52a06c2c283ecfa32
Key [14]:181715e5192efb2a64129668cf5d9dd4
round  8:eee3e8744a5f8896de95831ed837ffd5
Key [15]:83017c1434342d4290e961578790f451
Key [16]:2603532f365604646ff65803795ccce5
Key [17]:882f7c907b565ea58dae1c928a0dcf41
kc      :cc802aecc7312285912e90af6a1e1154
----------------------------------------
rand    :950e604e655ea3800fe3eb4a28918087
aco     :68f4f472b5586ac5850f5f74
key     :34e86915d20c485090a6977931f96df5
round  1:950e604e655ea3800fe3eb4a28918087
Key [ 1]:34e86915d20c485090a6977931f96df5
Key [ 2]:8de2595003f9928efaf37e5229935bdb
round  2:d46f5a04c967f55840f83d1cdb5f9afc
Key [ 3]:46f05ec979a97cb6ddf842ecc159c04a
Key [ 4]:b468f0190a0a83783521deae8178d071
round  3:e16edede9cb6297f32e1203e442ac73a
Key [ 5]:8a171624dedbd552356094daaadcf12a
Key [ 6]:3085e07c85e4b99313f6e0c837b5f819
round  4:805144e55e1ece96683d23366fc7d24b
Key [ 7]:fe45c27845169a66b679b2097d147715
Key [ 8]:44e2f0c35f64514e8bec66c5dc24b3ad
round  5:edbaf77af070bd22e9304398471042f1
Key [ 9]:0d534968f3803b6af447eaf964007e7b
Key [10]:f5499a32504d739ed0b3c547e84157ba
round  6:0dab1a4c846aef0b65b1498812a73b50
Key [11]:e17e8e456361c46298e6592a6311f3fb
Key [12]:ec6d14da05d60e8abac807646931711f
round  7:1e7793cac7f55a8ab48bd33bc9c649e0
Key [13]:2b53dde3d89e325e5ff808ed505706ae
Key [14]:41034e5c3fb0c0d4f445f0cf23be79b0
round  8:3723768baa78b6a23ade095d995404da
Key [15]:e2ca373d405a7abf22b494f28a6fd247
Key [16]:74e09c9068c0e8f1c6902d1b70537c30
Key [17]:767a7f1acf75c3585a55dd4a428b2119
round  1:39809afb773efd1b7510cd4cb7c49f34
Key [ 1]:1d0d48d485abddd3798b483a82a0f878
Key [ 2]:aed957e600a5aed5217984dd5fef6fd8
round  2:6436ddbabe92655c87a7d0c12ae5e5f6
Key [ 3]:fee00bb0de89b6ef0a289696a4faa884
Key [ 4]:33ce2f4411db4dd9b7c42cc586b8a2ba
round  3:cec690f7e0aa5f063062301e049a5cc5
added ->:f7462a0c97e85c1d4572fd52b35efbf1
```

*Sample Data*

```
Key [ 5]:b5116f5c6c29e05e4acb4d02a46a3318
Key [ 6]:ff4fa1f0f73d1a3c67bc2298abc768f9
round  4:dcdfe942e9f0163fc24a4718844b417d
Key [ 7]:5453650c0819e001e48331ad0e9076e0
Key [ 8]:b4ff8dda778e26c0dce08349b81c09a1
round  5:265a16b2f766afae396e7a98c189fda9
Key [ 9]:f638fa294427c6ed94300fd823b31d10
Key [10]:1ccfa0bd86a9879b17d4bc457e3e03d6
round  6:628576b5291d53d1eb8611c8624e863e
Key [11]:0eaee2ef4602ac9ca19e49d74a76d335
Key [12]:6e1062f10a16e0d378476da3943842e9
round  7:d7b9c2e9b2d5ea5c27019324cae882b3
Key [13]:40be960bd22c744c5b23024688e554b9
Key [14]:95c9902cb3c230b44d14ba909730d211
round  8:97fb6065498385e47eb3df6e2ca439dd
Key [15]:10d4b6e1d1d6798aa00aa2951e32d58d
Key [16]:c5d4b91444b83ee578004ab8876ba605
Key [17]:1663a4f98e2862eddd3ec2fb03dcc8a4
kc      :c1beafea6e747e304cf0bd7734b0a9e2
----------------------------------------
rand    :6a8ebcf5e6e471505be68d5eb8a3200c
aco     :658d791a9554b77c0b2f7b9f
key     :35cf77b333c294671d426fa79993a133
round  1:6a8ebcf5e6e471505be68d5eb8a3200c
Key [ 1]:35cf77b333c294671d426fa79993a133
Key [ 2]:c4524e53b95b4bf2d7b2f095f63545fd
round  2:ade94ec585db0d27e17474b58192c87a
Key [ 3]:c99776768c6e9f9dd3835c52cea8d18a
Key [ 4]:f1295db23823ba2792f21217fc01d23f
round  3:da8dc1a10241ef9e6e069267cd2c6825
Key [ 5]:9083db95a6955235bbfad8aeefec5f0b
Key [ 6]:8bab6bc253d0d0c7e0107feab728ff68
round  4:e6665ca0772ceecbc21222ff7be074f8
Key [ 7]:2fa1f4e7a4cf3ccd876ec30d194cf196
Key [ 8]:267364be247184d5337586a19df8bf84
round  5:a857a9326c9ae908f53fee511c5f4242
Key [ 9]:9aef21965b1a6fa83948d107026134c7
Key [10]:d2080c751def5dc0d8ea353cebf7b973
round  6:6678748a1b5f21ac05cf1b117a7c342f
Key [11]:d709a8ab70b0d5a2516900421024b81e
Key [12]:493e4843805f1058d605c8d1025f8a56
round  7:766c66fe9c460bb2ae39ec01e435f725
Key [13]:b1ed21b71daea03f49fe74b2c11fc02b
Key [14]:0e1ded7ebf23c72324a0165a698c65c7
round  8:396e0ff7b2b9b7a3b35c9810882c7596
Key [15]:b3bf4841dc92f440fde5f024f9ce8be9
Key [16]:1c69bc6c2994f4c84f72be8f6b188963
Key [17]:bb7b66286dd679a471e2792270f3bb4d
round  1:45654f2f26549675287200f07cb10ec9
Key [ 1]:1e2a5672e66529e4f427b0682a3a34b6
Key [ 2]:974944f1ce0037b1febcf61a2bc961a2
round  2:990cd869c534e76ed4f4af7b3bfbc6c8
```

*Sample Data*

```
Key [ 3]:8147631fb1ce95d624b480fc7389f6c4
Key [ 4]:6e90a2db33d284aa13135f3c032aa4f4
round  3:ceb662f875aa6b94e8192b5989abf975
added ->:8b1bb1d753fe01e1c08b2ba9f55c07bc
Key [ 5]:cbad246d24e36741c46401e6387a05f9
Key [ 6]:dcf52aaec5713110345a41342c566fc8
round  4:d4e000be5de78c0f56ff218f3c1df61b
Key [ 7]:8197537aa9d27e67d17c16b182c8ec65
Key [ 8]:d66e00e73d835927a307a3ed79d035d8
round  5:9a4603bdef954cfaade2052604bed4e4
Key [ 9]:71d46257ecc1022bcd312ce6c114d75c
Key [10]:f91212fa528379651fbd2c32890c5e5f
round  6:09a0fd197ab81eb933eece2fe0132dbb
Key [11]:283acc551591fadce821b02fb9491814
Key [12]:ca5f95688788e20d94822f162b5a3920
round  7:494f455a2e7a5db861ece816d4e363e4
Key [13]:ba574aef663c462d35399efb999d0e40
Key [14]:6267afc834513783fef1601955fe0628
round  8:37a819f91c8380fb7880e640e99ca947
Key [15]:fdcd9be5450eef0f8737e6838cd38e2b
Key [16]:8cfbd9b8056c6a1ce222b92b94319b38
Key [17]:4f64c1072c891c39eeb95e63318462e0
kc      :a3032b4df1cceba8adc1a04427224299
----------------------------------------
rand    :5ecd6d75db322c75b6afbd799cb18668
aco     :63f701c7013238bbf88714ee
key     :b9f90c53206792b1826838b435b87d4d
round  1:5ecd6d75db322c75b6afbd799cb18668
Key [ 1]:b9f90c53206792b1826838b435b87d4d
Key [ 2]:15f74bbbde4b9d1e08f858721f131669
round  2:72abb85fc80c15ec2b00d72873ef9ad4
Key [ 3]:ef7fb29f0b01f82706c7439cc52f2dab
Key [ 4]:3003a6aecdee06b9ac295cce30dcdb93
round  3:2f10bab93a0f73742183c68f712dfa24
Key [ 5]:5fcdbb3afdf7df06754c954fc6340254
Key [ 6]:ddaa90756635579573fe8ca1f93d4a38
round  4:183b145312fd99d5ad08e7ca4a52f04e
Key [ 7]:27ca8a7fc703aa61f6d7791fc19f704a
Key [ 8]:702029d8c6e42950762317e730ec5d18
round  5:cbad52d3a026b2e38b9ae6fefffecc32
Key [ 9]:ff15eaa3f73f4bc2a6ccfb9ca24ed9c5
Key [10]:034e745246cd2e2cfc3bda39531ca9c5
round  6:ce5f159d0a1acaacd9fb4643272033a7
Key [11]:0a4d8ff5673731c3dc8fe87e39a34b77
Key [12]:637592fab43a19ac0044a21afef455a2
round  7:8a49424a10c0bea5aba52dbbffcbcce8
Key [13]:6b3fde58f4f6438843cdbe92667622b8
Key [14]:a10bfa35013812f39bf2157f1c9fca4e
round  8:f5e12da0e93e26a5850251697ec0b917
Key [15]:2228fe5384e573f48fdd19ba91f1bf57
Key [16]:5f174db2bc88925c0fbc6b5485bafc08
Key [17]:28ff90bd0dc31ea2bb479feb7d8fe029
```

*Sample Data*

```
round  1:0c75eed2b54c1cfb9ff522daef94ed4d
Key [ 1]:a21ceb92d3c027326b4de775865fe8d0
Key [ 2]:26f64558a9f0a1652f765efd546f3208
round  2:48d537ac209a6aa07b70000016c602e8
Key [ 3]:e64f9ef630213260f1f79745a0102ae5
Key [ 4]:af6a59d7cebfd0182dcca9a537c4add8
round  3:8b6d517ac893743a401b3fb7911b64e1
added ->:87e23fa87ddf90c1df10616d7eaf51ac
Key [ 5]:9a6304428b45da128ab64c8805c32452
Key [ 6]:8af4d1e9d80cb73ec6b44e9b6e4f39d8
round  4:9f0512260a2f7a5067efc35bf1706831
Key [ 7]:79cc2d138606f0fca4e549c34a1e6d19
Key [ 8]:803dc5cdde0efdbee7a1342b2cd4d344
round  5:0cfd7856edfafac51f29e86365de6f57
Key [ 9]:e8fa996448e6b6459ab51e7be101325a
Key [10]:2acc7add7b294acb444cd933f0e74ec9
round  6:2f1fa34bf352dc77c0983a01e8b7d622
Key [11]:f57de39e42182efd6586b86a90c86bb1
Key [12]:e418dfd1bb22ebf1bfc309cd27f5266c
round  7:ee4f7a53849bf73a747065d35f3752b1
Key [13]:80a9959133856586370854db6e0470b3
Key [14]:f4c1bc2f764a0193749f5fc09011a1ae
round  8:8fec6f7249760ebf69e370e9a4b80a92
Key [15]:d036cef70d6470c3f52f1b5d25b0c29d
Key [16]:d0956af6b8700888a1cc88f07ad226dc
Key [17]:1ce8b39c4c7677373c30849a3ee08794
kc      :ea520cfc546b00eb7c3a6cea3ecb39ed

========================================
```

# SECURITY SPECIFICATION

*This document describes the specification of the security system which may be used at the link layer. The Encryption, Authentication and Key Generation schemes are specified. The requirements for the supporting process of random number generation are also specified.*


Ford Ex. 1052

# CONTENTS

**1 Security Overview ..... 1059**
1.1 Pausing Encryption and Role Switch ..... 1060
1.2 Change Connection Link Keys ..... 1060
1.3 Periodically Refreshing Encryption Keys ..... 1060

**2 Random Number Generation ..... 1061**

**3 Key Management ..... 1063**
3.1 Key Types ..... 1063
3.2 Key Generation and Initialization ..... 1065
3.2.1 Generation of initialization key, ..... 1066
3.2.2 Authentication ..... 1066
3.2.3 Generation of a unit key ..... 1066
3.2.4 Generation of a combination key ..... 1067
3.2.5 Generating the encryption key ..... 1068
3.2.6 Point-to-multipoint configuration ..... 1069
3.2.7 Modifying the link keys ..... 1069
3.2.8 Generating a master key ..... 1070

**4 Encryption ..... 1072**
4.1 Encryption Key Size Negotiation ..... 1073
4.2 Encryption of Broadcast Messages ..... 1073
4.3 Encryption Concept ..... 1074
4.4 Encryption Algorithm ..... 1075
4.4.1 The operation of the cipher ..... 1077
4.5 LFSR Initialization ..... 1078
4.6 Key Stream Sequence ..... 1081

**5 Authentication ..... 1082**
5.1 Repeated Attempts ..... 1084

**6 The Authentication And Key-Generating Functions ..... 1085**
6.1 The Authentication Function E1 ..... 1085
6.2 The Functions Ar and A'r ..... 1087
6.2.1 The round computations ..... 1087
6.2.2 The substitution boxes "e" and "l" ..... 1087
6.2.3 Key scheduling ..... 1088
6.3 E2-Key Generation Function for Authentication ..... 1089
6.4 E3-Key Generation Function for Encryption ..... 1091

**7 Secure Simple Pairing ..... 1092**
7.1 Phase 1: Public Key Exchange ..... 1093
7.2 Phase 2: Authentication Stage 1 ..... 1094

7.2.1 Authentication Stage 1: Numeric Comparison Protocol ..... 1094
7.2.2 Authentication Stage 1: Out of Band Protocol ..... 1096
7.2.3 Authentication Stage 1: Passkey Entry Protocol ..... 1098
7.3 Phase 3: Authentication Stage 2 ..... 1100
7.4 Phase 4: Link Key Calculation ..... 1101
7.5 Phase 5: LMP Authentication and Encryption ..... 1101
7.6 Elliptic Curve Definition ..... 1101
7.7 Cryptographic Function Definitions ..... 1102
7.7.1 The Simple Pairing Commitment Function f1 ..... 1102
7.7.2 The Simple Pairing Numeric Verification Function g ..... 1103
7.7.3 The Simple Pairing Key Derivation Function f2 ..... 1104
7.7.4 The Simple Pairing Check Function f3 ..... 1105
7.7.5 The Simple Pairing AMP Key Derivation Function h2 ..... 1106

**8 AMP Security ..... 1108**
8.1 Creation of the Initial Generic AMP Link Key ..... 1108
8.2 Creation of Dedicated AMP Link Keys ..... 1108
8.3 Debug Considerations ..... 1109

**9 List of Figures ..... 1111**

# 1 SECURITY OVERVIEW

Bluetooth wireless technology provides peer-to-peer communications over short distances. In order to provide usage protection and information confidentiality, the system provides security measures both at the application layer and the link layer. These measures are designed to be appropriate for a peer environment. This means that in each device, the authentication and encryption routines are implemented in the same way. Four different entities are used for maintaining security at the link layer: a Bluetooth device address, two secret keys, and a pseudo-random number that shall be regenerated for each new transaction. The four entities and their sizes are summarized in Table 1.1.

| Entity | Size |
|---|---|
| BD_ADDR | 48 bits |
| Private user key, authentication | 128 bits |
| Private user key, encryption configurable length (byte-wise) | 8-128 bits |
| RAND | 128 bits |

*Table 1.1: Entities used in authentication and encryption procedures.*

The Bluetooth device address (BD_ADDR) is the 48-bit address. The BD_ADDR can be obtained via user interactions, or, automatically, via an inquiry routine by a device.

The secret keys are derived during initialization and are never disclosed. The encryption key is derived from the authentication key during the authentication process. For the authentication algorithm, the size of the key used is always 128 bits. For the encryption algorithm, the key size may vary between 1 and 16 octets (8 - 128 bits). The size of the encryption key is configurable for two reasons. The first has to do with the many different requirements imposed on cryptographic algorithms in different countries – both with respect to export regulations and official attitudes towards privacy in general. The second reason is to facilitate a future upgrade path for the security without the need of a costly redesign of the algorithms and encryption hardware; increasing the effective key size is the simplest way to combat increased computing power at the opponent side.

The encryption key is entirely different from the authentication key (even though the latter is used when creating the former, as is described in Section 6.4 on page 1091). Each time encryption is activated, a new encryption key shall be generated. Thus, the lifetime of the encryption key does not necessarily correspond to the lifetime of the authentication key.

It is anticipated that the authentication key will be more static in its nature than the encryption key – once established, the particular application running on the device decides when, or if, to change it. To underline the fundamental impor-

tance of the authentication key to a specific link, it is often referred to as the link key.

The RAND is a pseudo-random number which can be derived from a random or pseudo-random process in the device. This is not a static parameter and will change frequently.

In the remainder of this chapter, the terms user and application are used interchangeably to designate the entity that is at either side.

## 1.1 PAUSING ENCRYPTION AND ROLE SWITCH

To perform a role switch on a connection encryption must be disabled or paused as the encryption is based off the master's clock and Bluetooth address information. Unfortunately, if the role switch is required, and encryption is turned off, the device on the other end of the link will not be aware of the reason for disabling encryption, and can therefore take two possible actions: send clear text data, or disconnect. Neither of these possible actions is desirable. When performing a role switch on an encrypted link, the role switch shall be performed as a single operation where possible. If this is not possible because the other device does not support the encryption pause feature, then when a device wishes to have an encrypted link, and encryption is disabled, then the device should not send any user data, and should not disconnect. If both devices support the encryption pause feature, then this procedure shall be used.

## 1.2 CHANGE CONNECTION LINK KEYS

It is possible to perform a change of connection link keys while a link is encrypted, but it is not possible to use the new link keys until encryption has been stopped and then restarted. As with role switches, disabling encryption and then re-enabling it again can cause user data to be sent in the clear or a disconnection to occur. The use of encryption pausing prevents this problem from occurring. On devices that do not support the encryption pause feature, when a device wishes to have an encrypted link, and encryption is disabled, then the device should not send any data, and should not disconnect. If both devices support the encryption pause feature, then this procedure shall be used.

## 1.3 PERIODICALLY REFRESHING ENCRYPTION KEYS

If both devices support the encryption pause feature, then the encryption keys shall be refreshed by the Link Manager at least once every $2^{28}$ Bluetooth Clocks (about 23.3 hours) if it is not refreshed by the Host or the remote Link Manager. To refresh an encryption key, the Host maybe use the Change Connection Link Key procedure or request an encryption key refresh. If the encryption key has not been refreshed before going stale, a device may disconnect the link.

# 2 RANDOM NUMBER GENERATION

Each device has a pseudo-random number generator. Pseudo-random numbers are used for many purposes within the security functions – for instance, for the challenge-response scheme, for generating authentication and encryption keys, nonces used in Simple Pairing, for Passkeys used in authentication.

$1/2^L$ A device shall use a pseudo random number generator compliant with [FIPS PUB 140-2] (http://csrc.nist.gov/publications/fips/fips140-2/fips1402annexc.pdf)

An example of a possible random generator is provided in [FIPS PUB 186-2] Appendix 3.1 (http://csrc.nist.gov/publications/fips/archives/fips186-2/fips186-2-change1.pdf), which can be used by replacing SHA-1 function with the SHA-256 function

The device shall use a seed with at least the minimum entropy required by the pseudo random number generator.

The random number generator shall be tested against the [FIPS SP800-22] (http://csrc.nist.gov/publications/nistpubs/800-22-rev1/SP800-22rev1.pdf). This encompasses the verification of the following statistical tests performed on the output of the PRNG as specified by the [FIPS SP800-22]:

1. The Frequency (Monobit) Test
2. Frequency Test within a Block
3. The Runs Test
4. Test for the Longest-Run-of-Ones in a Block
5. The Binary Matrix Rank Test
6. The Discrete Fourier Transform (Spectral) Test
7. The Non-overlapping Template Matching Test
8. The Overlapping Template Matching Test
9. Maurer's "Universal Statistical" Test
10. The Lempel-Ziv Compression Test
11. The Linear Complexity Test
12. The Serial Test
13. The Approximate Entropy Test
14. The Cumulative Sums (Cusums) Test
15. The Random Excursions Test
16. The Random Excursions Variant Test

These tests are part of standard statistical mathematical packages. Some test suites, like the Diehard test suite can be used to verify the compliance. Alternatively, other tools, such as the DieHarder (http://www.phy.duke.edu/~rgb/General/rand_rate.php) or the available NIST tools (http://csrc.nist.gov/groups/ST/

toolkit/random_number.html) and the corresponding recommendations (http://csrc.nist.gov/publications/nistpubs/800-22-rev1/SP800-22rev1.pdf) may also be used.

# 3 KEY MANAGEMENT

It is important that the encryption key size within a specific device cannot be set by the user – this should be a factory preset entity. In order to prevent the user from over-riding the permitted key size, the Bluetooth baseband processing shall not accept an encryption key given from higher software layers. Whenever a new encryption key is required, it shall be created as defined in Section 6.4 on page 1091.

Changing a link key shall also be done through the defined baseband procedures. Depending on what kind of link key it is, different approaches are required. The details are found in Section 3.2.7 on page 1069.

## 3.1 KEY TYPES

The link key is a 128-bit random number which is shared between two or more parties and is the base for all security transactions between these parties. The link key itself is used in the authentication routine. Moreover, the link key is used as one of the parameters when the encryption key is derived.

In the following, a session is defined as the time interval for which the device is a member of a particular piconet. Thus, the session terminates when the device disconnects from the piconet.

The link keys are either semi-permanent or temporary. A semi-permanent link key may be stored in non-volatile memory and may be used after the current session is terminated. Consequently, once a semi-permanent link key is defined, it may be used in the authentication of several subsequent connections between the devices sharing it. The designation semi-permanent is justified by the possibility of changing it. How to do this is described in Section 3.2.7 on page 1069.

The lifetime of a temporary link key is limited by the lifetime of the current session – it shall not be reused in a later session. Typically, in a point-to-multipoint configuration where the same information is to be distributed securely to several recipients, a common encryption key is useful. To achieve this, a special link key (denoted master key) may temporarily replace the current link keys. The details of this procedure are found in Section 3.2.6 on page 1069.

In the following, the current link key is the link key in use at the current moment. It can be semi-permanent or temporary. Thus, the current link key is used for all authentications and all generation of encryption keys in the on-going connection (session).

In order to accommodate different types of applications, four types of link keys have been defined:

- the combination key $K_{AB}$
- the unit key $K_A$
- the temporary key $K_{master}$
- the initialization key $K_{init}$

Note: the use of unit keys is deprecated since it is implicitly insecure.

In addition to these keys there is an encryption key, denoted $K_c$. This key is derived from the current link key. Whenever encryption is activated by an LM command, the encryption key shall be changed automatically. The purpose of separating the authentication key and encryption key is to facilitate the use of a shorter encryption key without weakening the strength of the authentication procedure. There are no governmental restrictions on the strength of authentication algorithms. However, in some countries, such restrictions exist on the strength of encryption algorithms.

The combination key $K_{AB}$ and the unit key $K_A$ are functionally indistinguishable; the difference is in the way they are generated. The unit key $K_A$ is generated in, and therefore dependent on, a single device A. The unit key shall be generated once at installation of the device; thereafter, it is very rarely changed. The combination key is derived from information in both devices A and B, and is therefore always dependent on two devices. The combination key is derived for each new combination of two devices.

It depends on the application or the device whether a unit key or a combination key is used. Devices which have little memory to store keys, or are installed in equipment that will be accessible to a large group of users, should use their own unit key. In that case, they only have to store a single key. Applications that require a higher security level should use the combination keys. These applications will require more memory since a combination key for each link to a different device has to be stored.

The master key, $K_{master}$, shall only be used during the current session. It shall only replace the original link key temporarily. For example, this may be utilized when a master wants to reach more than two devices simultaneously using the same encryption key, see Section 3.2.6 on page 1069.

The initialization key, $K_{init}$, shall be used as the link key during the initialization process when no combination or unit keys have been defined and exchanged yet or when a link key has been lost. The initialization key protects the transfer of initialization parameters. The key is derived from a random number, an L-octet PIN code, and a BD_ADDR. This key shall only be used during initialization.

The PIN may be a fixed number provided with the device (for example when there is no user interface as in a PSTN plug). Alternatively, the PIN can be selected by the user, and then entered in both devices that are to be matched. The latter procedure should be used when both devices have a user interface, for example a phone and a laptop. Entering a PIN in both devices is more secure than using a fixed PIN in one of the devices, and should be used whenever possible. Even if a fixed PIN is used, it shall be possible to change the PIN; this is in order to prevent re-initialization by users who once obtained the PIN. If no PIN is available, a default value of zero may be used. The length of this default PIN is one byte, PIN (default) = 0x00. This default PIN may be provided by the host.

For many applications the PIN code will be a relatively short string of numbers. Typically, it may consist of only four decimal digits. Even though this gives sufficient security in many cases, there exist countless other, more sensitive, situations where this is not reliable enough. Therefore, the PIN code may be chosen to be any length from 1 to 16 octets. For the longer lengths, the devices exchanging PIN codes may not use mechanical (i.e. human) interaction, but rather may use software at the application layer. For example, this can be a Diffie-Hellman key agreement, where the exchanged key is passed on to the $K_{init}$ generation process in both devices, just as in the case of a shorter PIN code.

## 3.2 KEY GENERATION AND INITIALIZATION

The link keys must be generated and distributed among the devices in order to be used in the authentication procedure. Since the link keys shall be secret, they shall not be obtainable through an inquiry routine in the same way as the Bluetooth device addresses. The exchange of the keys takes place during an initialization phase which shall be carried out separately for each two devices that are using authentication and encryption. The initialization procedures consist of the following five parts:

- generation of an initialization key
- generation of link key
- link key exchange
- authentication
- generation of encryption key in each device (optional)

After the initialization procedure, the devices can proceed to communicate, or the link can be disconnected. If encryption is implemented, the $E_0$ algorithm shall be used with the proper encryption key derived from the current link key. For any new connection established between devices A and B, they should use the common link key for authentication, instead of once more deriving $K_{init}$ from the PIN. A new encryption key derived from that particular link key shall be created next time encryption is activated.

If no link key is available, the LM shall automatically start an initialization procedure.

### 3.2.1 Generation of initialization key, $K_{init}$

A link key is used temporarily during initialization, the initialization key $K_{init}$. This key shall be derived by the $E_{22}$ algorithm from a BD_ADDR, a PIN code, the length of the PIN (in octets), and a random number IN_RAND. The principle is depicted in Figure 6.4 on page 1091. The 128-bit output from $E_{22}$ shall be used for key exchange during the generation of a link key. When the devices have performed the link key exchange, the initialization key shall be discarded.

When the initialization key is generated, the PIN is augmented with the BD_ADDR. If one device has a fixed PIN the BD_ADDR of the other device shall be used. If both devices have a variable PIN the BD_ADDR of the device that received IN_RAND shall be used. If both devices have a fixed PIN they cannot be paired. Since the maximum length of the PIN used in the algorithm cannot exceed 16 octets, it is possible that not all octets of BD_ADDR will be used. This procedure ensures that $K_{init}$ depends on the identity of the device with a variable PIN. A fraudulent device may try to test a large number of PINs by claiming another BD_ADDR each time. It is the application's responsibility to take countermeasures against this threat. If the device address is kept fixed, the waiting interval before the next try may be increased exponentially (see Section 5.1 on page 1084).

The details of the $E_{22}$ algorithm can be found in Section 6.3 on page 1089.

### 3.2.2 Authentication

The authentication procedure shall be carried out as described in Section 5 on page 1082. During each authentication, a new $AU\_RAND_A$ shall be issued.

Mutual authentication is achieved by first performing the authentication procedure in one direction and then immediately performing the authentication procedure in the opposite direction.

As a side effect of a successful authentication procedure an auxiliary parameter, the Authenticated Ciphering Offset (ACO), will be computed. The ACO shall be used for ciphering key generation as described in Section 3.2.5 on page 1068.

The claimant/verifier status is determined by the LM.

### 3.2.3 Generation of a unit key

A unit key $K_A$ shall be generated when the device is in operation for the first time; i.e. not during each initialization. The unit key shall be generated by the $E_{21}$ algorithm as described in Section 6.3 on page 1089. Once created, the unit key should be stored in non-volatile memory and very rarely changed. If after initial-

ization the unit key is changed, any previously initialized devices will possess a wrong link key. At initialization, the application must determine which of the two parties will provide the unit key as the link key. Typically, this will be the device with restricted memory capabilities, since this device only has to remember its own unit key. The unit key shall be transferred to the other party and then stored as the link key for that particular party. So, for example in Figure 3.1 on page 1067, the unit key of device A, $K_A$, is being used as the link key for the connection A-B; device A sends the unit key $K_A$ to device B; device B will store $K_A$ as the link key $K_{BA}$. For another initialization, for example with device C, device A will reuse its unit key $K_A$, whereas device C stores it as $K_{CA}$.

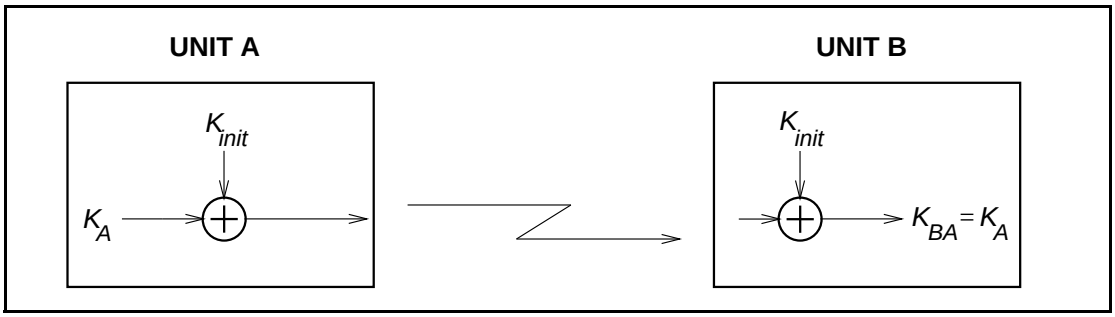



*Figure 3.1: Generation of unit key. When the unit key has been exchanged, the initialization key is discarded in both devices.*

### 3.2.4 Generation of a combination key

To use a combination key, it is first generated during the initialization procedure. The combination key is the combination of two numbers generated in device A and B, respectively. First, each device shall generate a random number, $LK\_RAND_A$ and $LK\_RAND_B$. Then, utilizing $E_{21}$ with the random number and their own BD_ADDRs, the two random numbers

$$LK\_K_A = E_{21}(LK\_RAND_A, BD\_ADDR_A) \quad \text{(EQ 1)}$$

and

$$LK\_K_B = E_{21}(LK\_RAND_B, BD\_ADDR_B), \quad \text{(EQ 2)}$$

shall be created in device A and device B, respectively. These numbers constitute the devices' contribution to the combination key that is to be created. Then, the two random numbers $LK\_RAND_A$ and $LK\_RAND_B$ shall be exchanged securely by XORing with the current link key, $K$. Thus, device A shall send $K \oplus LK\_RAND_A$ to device B, while device B shall send $K \oplus LK\_RAND_B$ to device A. If this is done during the initialization phase the link key $K = K_{init}$.

When the random numbers $LK\_RAND_A$ and $LK\_RAND_B$ have been mutually exchanged, each device shall recalculate the other device's contribution to the combination key. This is possible since each device knows the Bluetooth

device address of the other device. Thus, device A shall calculate (EQ 2) on page 1067 and device B shall calculate (EQ 1) on page 1067. After this, both devices shall combine the two numbers to generate the 128-bit link key. The combining operation is a simple bitwise modulo-2 addition (i.e. XOR). The result shall be stored in device A as the link key $K_{AB}$ and in device B as the link key $K_{BA}$. When both devices have derived the new combination key, a mutual authentication procedure shall be initiated to confirm the success of the transaction. The old link key shall be discarded after a successful exchange of a new combination key. The message flow between master and slave and the principle for creating the combination key is depicted in Figure 3.2 on page 1068.

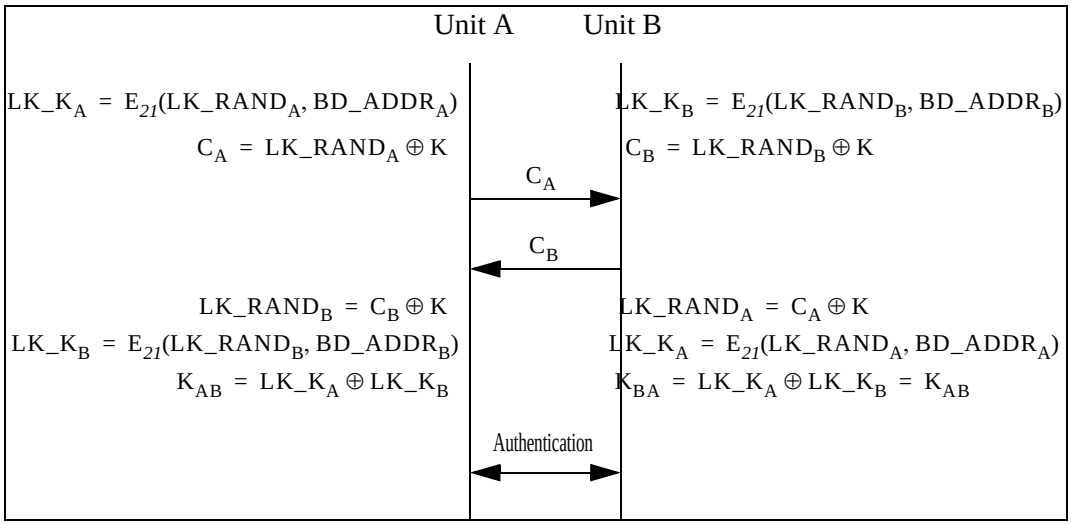



*Figure 3.2: Generating a combination key. The old link key (K) is discarded after the exchange of a new combination key has succeeded*

### 3.2.5 Generating the encryption key

The encryption key, $K_C$, is derived by algorithm $E_3$ from the current link key, a 96-bit Ciphering OFfset number (COF), and a 128-bit random number. The COF is determined in one of two ways. If the current link key is a master key, then COF shall be derived from the master BD_ADDR. Otherwise the value of COF shall be set to the value of ACO as computed during the authentication procedure. Therefore:[1]

$$\text{COF} = \begin{cases} \text{BD\_ADDR} \cup \text{BD\_ADDR}, & \text{if link key is a master key} \\ \text{ACO}, & \text{otherwise.} \end{cases} \quad \text{(EQ 3)}$$

There is an explicit call of $E_3$ when the LM activates encryption. Consequently, the encryption key is automatically changed each time the device enters

1. $x \cup y$ denotes the concatenation of $x$ and $y$.

encryption mode. The details of the key generating function $E_3$ can be found in Section 6.4 on page 1091.

### 3.2.6 Point-to-multipoint configuration

It is possible for the master to use separate encryption keys for each slave in a point-to-multipoint configuration with ciphering activated. Then, if the application requires more than one slave to listen to the same payload, each slave must be addressed individually. This can cause unwanted capacity loss for the piconet. Moreover, a slave might not be capable of switching between two or more encryption keys in real time (e.g., after looking at the LT_ADDR in the header). Thus, the master cannot use different encryption keys for broadcast messages and individually addressed traffic. Therefore, the master may tell several slave devices to use a common link key (and, hence, indirectly also to use a common encryption key) and may then broadcast the information encrypted. For many applications, this key is only of temporary interest. In the following discussion, this key is denoted by $K_{master}$.

The transfer of necessary parameters shall be protected by the routine described in Section 3.2.8 on page 1070. After the confirmation of successful reception in each slave, the master shall issue a command to the slaves to replace their respective current link key by the new (temporary) master key. Before encryption can be activated, the master shall also generate and distribute a common EN_RAND to all participating slaves. Using this random number and the newly derived master key, each slave shall generate a new encryption key.

Note that the master must negotiate the encryption key length to use individually with each slave that will use the master key. If the master has already negotiated with some of these slaves, it has knowledge of the sizes that can be accepted. There may be situations where the permitted key lengths of some devices are incompatible. In that case, the master must exclude the limiting device from the group.

When all slaves have received the necessary data, the master can communicate information on the piconet securely using the encryption key derived from the new temporary link key. Each slave in possession of the master key can eavesdrop on all encrypted traffic, not only the traffic intended for itself. The master may tell all participants to fall back to their old link keys simultaneously.

### 3.2.7 Modifying the link keys

A link key based on a unit key can be changed. The unit key is created once during first use. Typically, the link key should be changed rather than the unit key, as several devices may share the same unit key as link key (e.g. a printer whose unit key is distributed to all users using the printer's unit key as link key). Changing the unit key will require re-initialization of all devices connecting.

Changing the unit key can be justified in some circumstances, e.g. to deny access to all previously allowed devices.

If the key change concerns combination keys, then the procedure is straightforward. The change procedure is identical to the procedure described in Figure 3.2 on page 1068, using the current value of the combination key as link key. This procedure can be carried out at any time after the authentication and encryption start. Since the combination key corresponds to a single link, it can be modified each time this link is established. This will improve the security of the system since then old keys lose their validity after each session.

Starting up an entirely new initialization procedure is also possible. In that case, user interaction is necessary since a PIN will be required in the authentication and encryption procedures.

### 3.2.8 Generating a master key

The key-change routines described so far are semi-permanent. To create the master link key, which can replace the current link key during a session (see Section 3.2.6 on page 1069), other means are needed. First, the master shall create a new link key from two 128-bit random numbers, RAND1 and RAND2. This shall be done by

$$K_{master} = E_{22}(RAND1, RAND2, 16). \quad \text{(EQ 4)}$$

This key is a 128-bit random number. The reason for using the output of $E_{22}$ and not directly choosing a random number as the key, is to avoid possible problems with degraded randomness due to a poor implementation of the random number generator within the device.

Then, a third random number, RAND, shall be transmitted to the slave. Using $E_{22}$ with the current link key and RAND as inputs, both the master and the slave shall compute a 128-bit overlay. The master shall send the bitwise XOR of the overlay and the new link key to the slave. The slave, who knows the overlay, shall recalculate $K_{master}$. To confirm the success of this transaction, the devices shall perform a mutual authentication procedure using the new link key. This procedure shall then be repeated for each slave that receives the new link key. The ACO values from the authentications shall not replace the current ACO, as this ACO is needed to (re)compute a ciphering key when the master falls back to the previous (non-temporary) link key.

The master activates encryption by an LM command. Before activating encryption, the master shall ensure that all slaves receive the same random number, EN_RAND, since the encryption key is derived through the means of $E_3$ individually in all participating devices. Each slave shall compute a new encryption key as follows:

$$K_C = E_3(K_{master}, EN\_RAND, COF) \quad \text{(EQ 5)}$$

where the value of COF shall be derived from the master's BD_ADDR as specified by equation (EQ 3) on page 1068. The details of the encryption key generating function are described in Section 6.4 on page 1091. The message flow between the master and the slave when generating the master key is depicted in Figure 3.3. Note that in this case the ACO produced during the authentication is not used when computing the ciphering key.

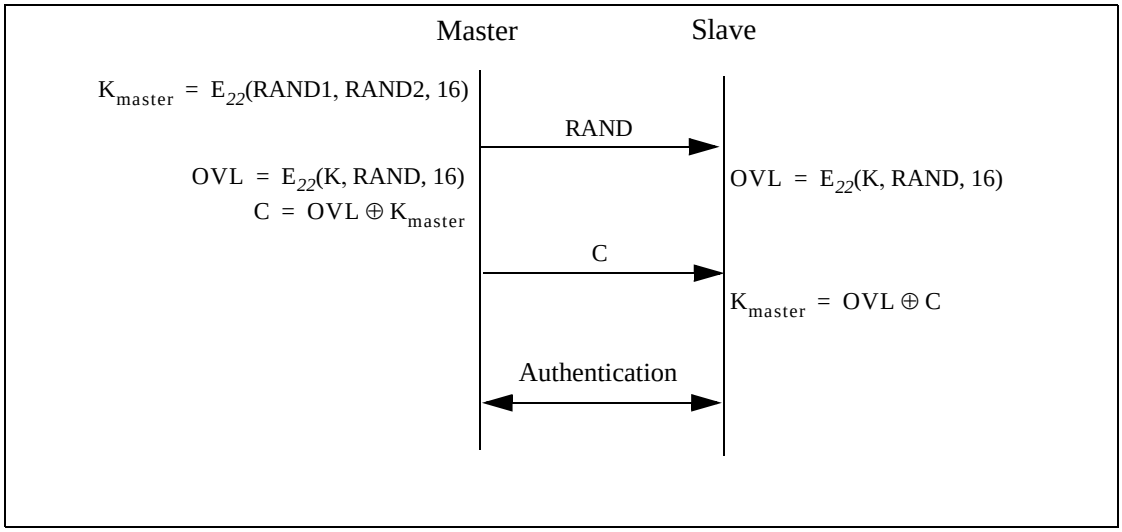



*Figure 3.3: Master link key distribution and computation of the corresponding encryption key.*

# 4 ENCRYPTION

User information can be protected by encryption of the packet payload; the access code and the packet header shall never be encrypted. The encryption of the payload shall be carried out with a stream cipher called $E_0$ that shall be re-synchronized for every payload. The overall principle is shown in Figure 4.1 on page 1072.

The stream cipher system $E_0$ shall consist of three parts:

- the first part performs initialization (generation of the payload key). The payload key generator shall combine the input bits in an appropriate order and shall shift them into the four LFSRs used in the key stream generator.
- the second part generates the key stream bits and shall use a method derived from the summation stream cipher generator attributable to Massey and Rueppel. The second part is the main part of the cipher system, as it will also be used for initialization.
- the third part performs encryption and decryption.

The Massey and Rueppel method has been thoroughly investigated, and there exist good estimates of its strength with respect to presently known methods for cryptanalysis. Although the summation generator has weaknesses that can be used in correlation attacks, the high re-synchronization frequency will disrupt such attacks.

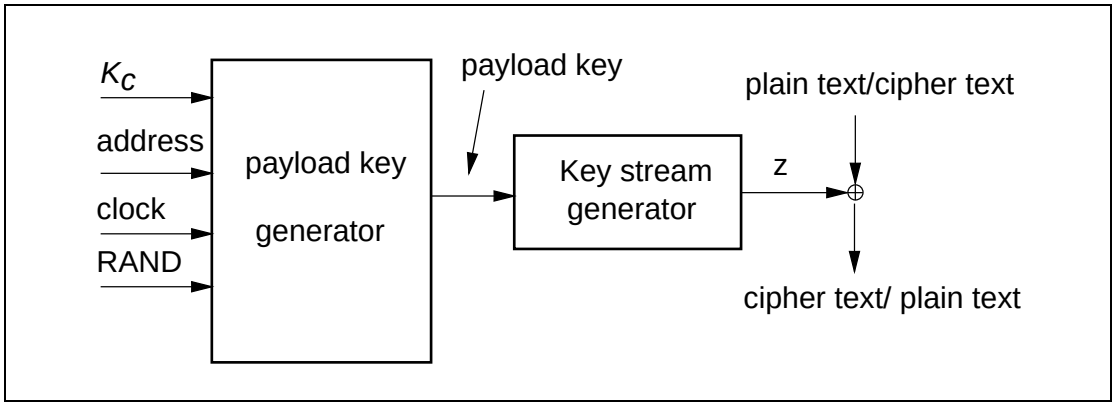



*Figure 4.1: Stream ciphering for Bluetooth with $E_0$.*

## 4.1 ENCRYPTION KEY SIZE NEGOTIATION

Each device implementing the baseband specification shall have a parameter defining the maximal allowed key length, $L_{max}$, $1 \leq L_{max} \leq 16$ (number of octets in the key). For each application using encryption, a number $L_{min}$ shall be defined indicating the smallest acceptable key size for that particular application. Before generating the encryption key, the devices involved shall negotiate to decide the key size to use.

The master shall send a suggested value, $L_{sug}^{(M)}$, to the slave. Initially, the suggested value shall be set to $L_{max}^{(M)}$. If $L_{min}^{(S)} \leq L_{sug}^{(M)}$, and, the slave supports the suggested length, the slave shall acknowledge and this value shall be the length of the encryption key for this link. However, if both conditions are not fulfilled, the slave shall send a new proposal, $L_{sug}^{(S)} < L_{sug}^{(M)}$, to the master. This value shall be the largest among all supported lengths less than the previous master suggestion. Then, the master shall perform the corresponding test on the slave suggestion. This procedure shall be repeated until a key length agreement is reached, or, one device aborts the negotiation. An abort may be caused by lack of support for $L_{sug}$ and all smaller key lengths, or if $L_{sug} < L_{min}$ in one of the devices. In case of an abort link encryption cannot be employed.

The possibility of a failure in setting up a secure link is an unavoidable consequence of letting the application decide whether to accept or reject a suggested key size. However, this is a necessary precaution. Otherwise a fraudulent device could enforce a weak protection on a link by claiming a small maximum key size.

## 4.2 ENCRYPTION OF BROADCAST MESSAGES

There may be three settings for the baseband regarding encryption:

1. No encryption.
   This is the default setting. No messages are encrypted
2. Point-to-point only encryption.
   Broadcast messages are not encrypted. This may be enabled either during the connection establishment procedure or after the connection has been established.
3. Point-to-point and broadcast encryption.
   All messages are encrypted. This may be enabled after the connection has been established only. This setting should not be enabled unless all affected links share the same master link key as well as the same EN_RAND value, both used in generating the encryption key.

## 4.3 ENCRYPTION CONCEPT

| Broadcast traffic | Individually addressed traffic |
|---|---|
| No encryption | No encryption |
| No encryption | Encryption, $K_{master}$ |
| Encryption, $K_{master}$ | Encryption, $K_{master}$ |

*Table 4.1: Possible encryption modes for a slave in possession of a master key.*

For the encryption routine, a stream cipher algorithm is used in which ciphering bits are bit-wise modulo-2 added to the data stream to be sent over the air interface. The payload is ciphered after the CRC bits are appended, but, prior to the FEC encoding.

Each packet payload shall be ciphered separately. The cipher algorithm $E_0$ uses the master Bluetooth device address (BD_ADDR), 26 bits of the master real-time clock (CLK $_{26\text{-}1}$) and the encryption key $K_C$ as input, see Figure 4.2 on page 1075 (where it is assumed that device A is the master).

The encryption key $K_C$ is derived from the current link key, COF, and a random number, $EN\_RAND_A$ (see Section 6.4 on page 1091). The random number shall be issued by the master before entering encryption mode. Note that $EN\_RAND_A$ is publicly known since it is transmitted as plain text over the air.

Within the $E_0$ algorithm, the encryption key $K_C$ is modified into another key denoted $K'_C$. The maximum effective size of this key shall be factory preset and may be set to any multiple of eight between one and sixteen (8-128 bits). The procedure for deriving the key is described in Section 4.5 on page 1078.

The real-time clock is incremented for each slot. The $E_0$ algorithm shall be re-initialized at the start of each new packet (i.e. for Master-to-Slave as well as for Slave-to-Master transmission). By using $CLK_{26\text{-}1}$ at least one bit is changed between two transmissions. Thus, a new keystream is generated after each re-initialization. For packets covering more than a single slot, the Bluetooth clock as found in the first slot shall be used for the entire packet.

The encryption algorithm $E_0$ generates a binary keystream, $K_{cipher}$, which shall be modulo-2 added to the data to be encrypted. The cipher is symmetric; decryption shall be performed in exactly the same way using the same key as used for encryption.

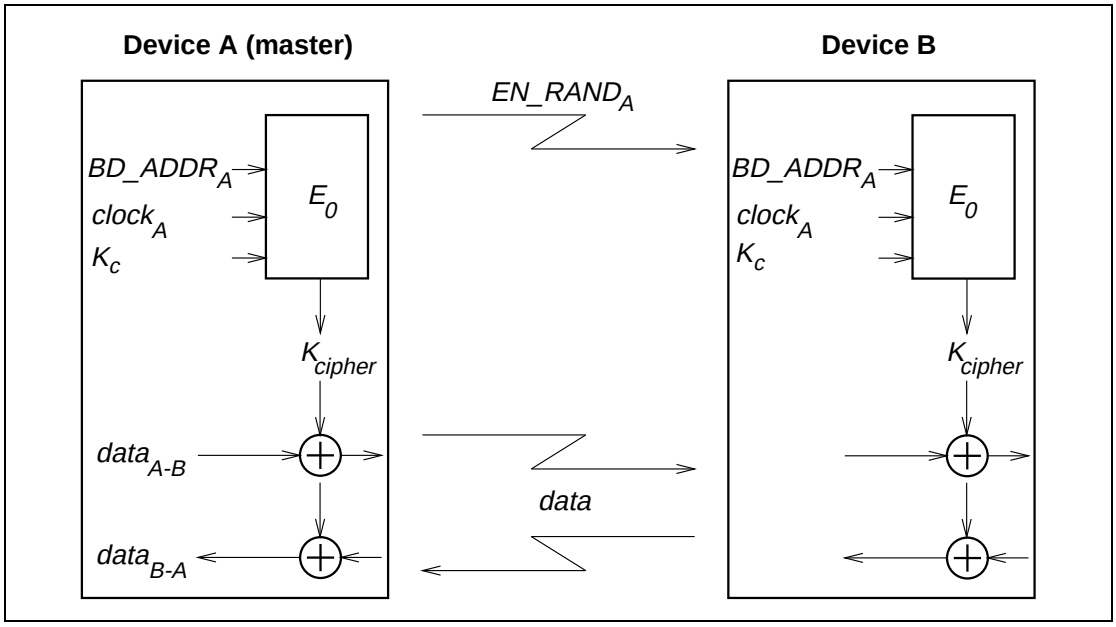



*Figure 4.2: Functional description of the encryption procedure*

## 4.4 ENCRYPTION ALGORITHM

The system uses linear feedback shift registers (LFSRs) whose output is combined by a simple finite state machine (called the summation combiner) with 16 states. The output of this state machine is the key stream sequence, or, during initialization phase, the randomized initial start value. The algorithm uses an encryption key $K_C$, a 48-bit Bluetooth address, the master clock bits $CLK_{26\text{-}1}$, and a 128-bit RAND value. Figure 4.3 on page 1076 shows the setup.

There are four LFSRs ($LFSR_1$,...,$LFSR_4$) of lengths $L_1 = 25$, $L_2 = 31$, $L_3 = 33$, and, $L_4 = 39$, with feedback polynomials as specified in Table 4.2 on page 1076. The total length of the registers is 128. These polynomials are all primitive. The Hamming weight of all the feedback polynomials is chosen to be five – a reasonable trade-off between reducing the number of required XOR gates in the hardware implementation and obtaining good statistical properties of the generated sequences.

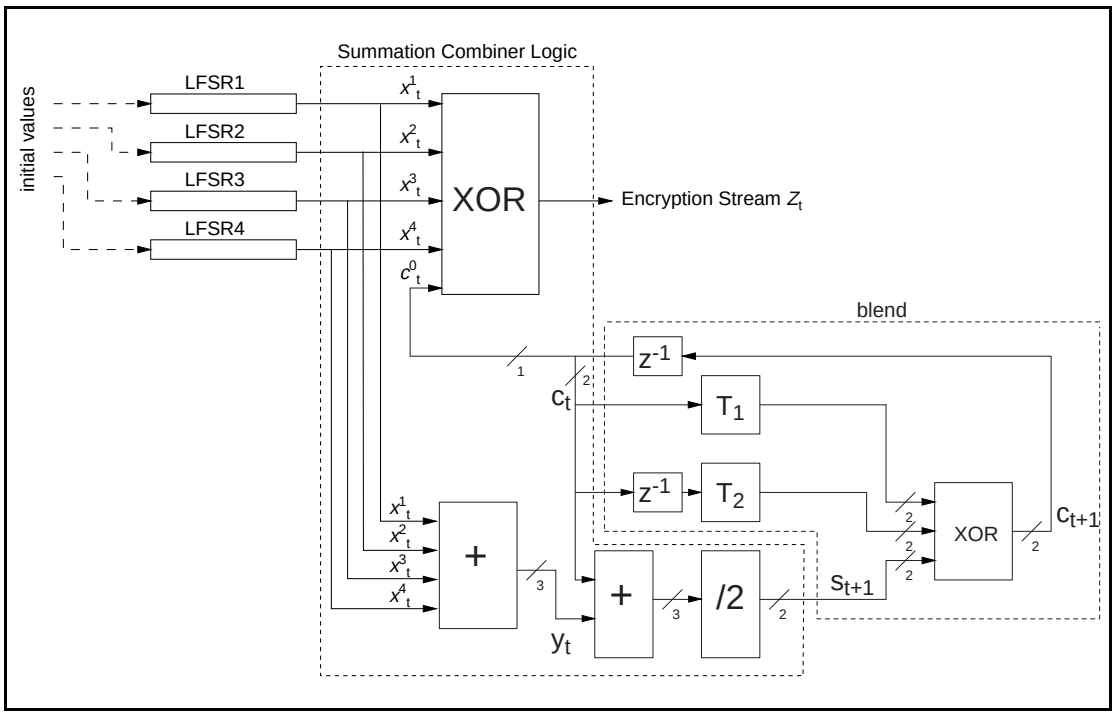



*Figure 4.3: Concept of the encryption engine.*

| i | $L_i$ | **feedback** $f_i(t)$ | **weight** |
|---|---|---|---|
| 1 | 25 | $t^{25} + t^{20} + t^{12} + t^8 + 1$ | 5 |
| 2 | 31 | $t^{31} + t^{24} + t^{16} + t^{12} + 1$ | 5 |
| 3 | 33 | $t^{33} + t^{28} + t^{24} + t^4 + 1$ | 5 |
| 4 | 39 | $t^{39} + t^{36} + t^{28} + t^4 + 1$ | 5 |

*Table 4.2: The four primitive feedback polynomials.*

Let $x_t^i$ denote the $t^{th}$ symbol of $LFSR_i$. The value $y_t$ is derived from the four-tuple $x_t^1, \ldots, x_t^4$ using the following equation:

$$y_t = \sum_{i=1}^{4} x_t^i, \qquad \text{(EQ 6)}$$

where the sum is over the integers. Thus $y_t$ can take the values 0,1,2,3, or 4. The output of the summation generator is obtained by the following equations:

$$z_t = x_t^1 \oplus x_t^2 \oplus x_t^3 \oplus x_t^4 \oplus c_t^0 \in \{0, 1\}, \qquad \text{(EQ 7)}$$

$$s_{t+1} = (s^1_{t+1}, s^0_{t+1}) = \left\lfloor \frac{y_t + c_t}{2} \right\rfloor \in \{0, 1, 2, 3\}, \quad \text{(EQ 8)}$$

$$c_{t+1} = (c^1_{t+1}, c^0_{t+1}) = s_{t+1} \oplus T_1[c_t] \oplus T_2[c_{t-1}], \quad \text{(EQ 9)}$$

where $T_1[.]$ and $T_2[.]$ are two different linear bijections over GF(4). Suppose GF(4) is generated by the irreducible polynomial $x^2 + x + 1$, and let $\alpha$ be a zero of this polynomial in GF(4). The mappings $T_1$ and $T_2$ are now defined as:

$$T_1: \mathrm{GF}(4) \rightarrow \mathrm{GF}(4)$$
$$x \mapsto x$$
$$T_2: \mathrm{GF}(4) \rightarrow \mathrm{GF}(4)$$
$$x \mapsto (\alpha + 1)x.$$

The elements of GF(4) can be written as binary vectors. This is summarized in Table 4.3.

| x | $T_1[x]$ | $T_2[x]$ |
|---|---|---|
| 00 | 00 | 00 |
| 01 | 01 | 11 |
| 10 | 10 | 01 |
| 11 | 11 | 10 |

*Table 4.3: The mappings $T_1$ and $T_2$.*

Since the mappings are linear, they can be implemented using XOR gates; i.e.

$$T_1: \quad (x_1, x_0) \mapsto (x_1, x_0),$$
$$T_2: \quad (x_1, x_0) \mapsto (x_0, x_1 \oplus x_0).$$

### 4.4.1 The operation of the cipher

Figure 4.4 on page 1078 gives an overview of the operation in time. The encryption algorithm shall run through the initialization phase before the start of transmission or reception of a new packet. Thus, for multislot packets the cipher is initialized using the clock value of the first slot in the multislot sequence.

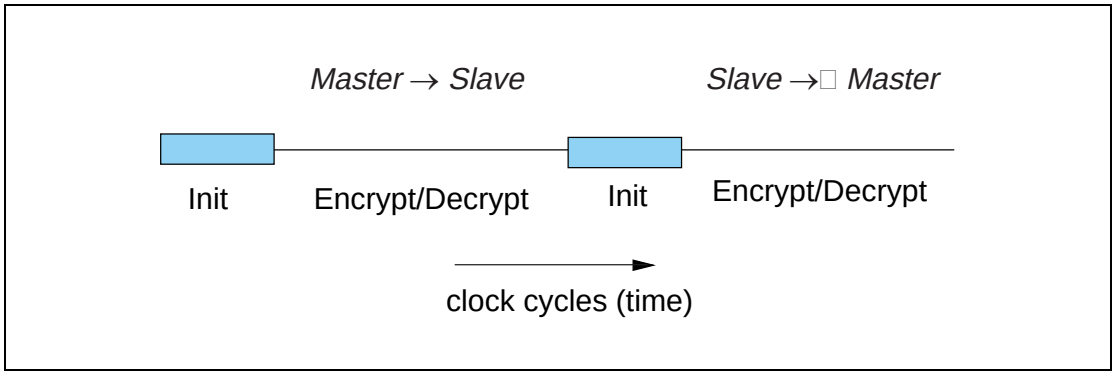



*Figure 4.4: Overview of the operation of the encryption engine. Between each start of a packet (TX or RX), the LFSRs are re-initialized.*

## 4.5 LFSR INITIALIZATION

The key stream generator is loaded with an initial value for the four LFSRs (in total 128 bits) and the 4 bits that specify the values of $c_0$ and $c_{-1}$. The 132 bit initial value is derived from four inputs by using the key stream generator. The input parameters are the key $K_C$, a 128-bit random number RAND, a 48-bit Bluetooth device address, and the 26 master clock bits $CLK_{26\text{-}1}$.

The effective length of the encryption key may vary between 8 and 128 bits. Note that the actual key length as obtained from $E_3$ is 128 bits. Then, within $E_0$, the key length may be reduced by a modulo operation between $K_C$ and a polynomial of desired degree. After reduction, the result is encoded with a block code in order to distribute the starting states more uniformly. The operation shall be as defined in (EQ 10) on page 1079.

When the encryption key has been created the LFSRs are loaded with their initial values. Then, 200 stream cipher bits are created by operating the generator. Of these bits, the last 128 are fed back into the key stream generator as an initial value of the four LFSRs. The values of $c_t$ and $c_{t-1}$ are kept. From this point on, when clocked the generator produces the encryption (decryption) sequence which is bitwise XORed to the transmitted (received) payload data.

In the following, octet $i$ of a binary sequence $X$ is $X[i]$. Bit 0 of $X$ is the LSB. Then, the LSB of $X[i]$ corresponds to bit $8i$ of the sequence $X$, the MSB of $X[i]$ is bit $8i + 7$ of $X$. For instance, bit 24 of the Bluetooth device address is the LSB of BD_ADDR[3].

The details of the initialization shall be as follows:

1. Create the encryption key to use from the 128-bit secret key $K_C$ and the 128-bit publicly known EN_RAND. Let $L$, $1 \le L \le 16$, be the

effective key length in number of octets. The resulting encryption key is $K'_C$:

$$K'_C(x) = g_2^{(L)}(x)(K_C(x) \bmod g_1^{(L)}(x)), \qquad \text{(EQ 10)}$$

where $\deg(g_1^{(L)}(x)) = 8L$ and $\deg(g_2^{(L)}(x)) \leq 128 - 8L$. The polynomials are defined in Table 4.4.

2. Shift the 3 inputs $K'_C$, the Bluetooth device address, the clock, and the six-bit constant 111001 into the LFSRs. In total, 208 bits are shifted in:

   a) Open all switches shown in Figure 4.5 on page 1080;

   b) Arrange inputs bits as shown in Figure 4.5; Set the content of all shift register elements to zero. Set $t = 0$.

   c) Start shifting bits into the LFSRs. The rightmost bit at each level of Figure 4.5 is the first bit to enter the corresponding LFSR.

   d) When the first input bit at level i reaches the rightmost position of $LFSR_i$, close the switch of this LFSR.

   e) At $t = 39$ (when the switch of $LFSR_4$ is closed), reset both blend registers $c_{39} = c_{39-1} = 0$; Up to this point, the content of $c_t$ and $c_{t-1}$ has been of no concern. However, their content will now be used in computing the output sequence.

   f) From now on output symbols are generated. The remaining input bits are continuously shifted into their corresponding shift registers. When the last bit has been shifted in, the shift register is clocked with input = 0;

      Note: When finished, $LFSR_1$ has effectively clocked 30 times with feedback closed, $LFSR_2$ has clocked 24 times, $LFSR_3$ has clocked 22 times, and $LFSR_4$ has effectively clocked 16 times with feedback closed.

3. To mix initial data, continue to clock until 200 symbols have been produced with all switches closed ($t = 239$);

4. Keep blend registers $c_t$ and $c_{t-1}$, make a parallel load of the last 128 generated bits into the LFSRs according to Figure 4.6 at $t = 240$;

After the parallel load in item 4, the blend register contents shall be updated for each subsequent clock.

| L | deg | $g_1^{(L)}$ | deg | $g_2^{(L)}$ |
|---|---|---|---|---|
| 1 | [8] | 00000000 00000000 00000000 0000011d | [119] | 00e275a0 abd218d4 cf928b9b bf6cb08f |
| 2 | [16] | 00000000 00000000 00000000 0001003f | [112] | 0001e3f6 3d7659b3 7f18c258 cff6efef |
| 3 | [24] | 00000000 00000000 00000000 010000db | [104] | 000001be f66c6c3a b1030a5a 1919808b |
| 4 | [32] | 00000000 00000000 00000001 000000af | [96] | 00000001 6ab89969 de17467f d3736ad9 |

*Table 4.4: Polynomials used when creating $K'_c$. .* [1]

| L | deg | $g_1^{(L)}$ | deg | $g_2^{(L)}$ |
|---|---|---|---|---|
| 5 | [40] | 00000000 00000000 00000100 00000039 | [88] | 00000000 01630632 91da50ec 55715247 |
| 6 | [48] | 00000000 00000000 00010000 00000291 | [77] | 00000000 00002c93 52aa6cc0 54468311 |
| 7 | [56] | 00000000 00000000 01000000 00000095 | [71] | 00000000 000000b3 f7fffce2 79f3a073 |
| 8 | [64] | 00000000 00000001 00000000 0000001b | [63] | 00000000 00000000 a1ab815b c7ec8025 |
| 9 | [72] | 00000000 00000100 00000000 00000609 | [49] | 00000000 00000000 0002c980 11d8b04d |
| 10 | [80] | 00000000 00010000 00000000 00000215 | [42] | 00000000 00000000 0000058e 24f9a4bb |
| 11 | [88] | 00000000 01000000 00000000 0000013b | [35] | 00000000 00000000 0000000c a76024d7 |
| 12 | [96] | 00000001 00000000 00000000 000000dd | [28] | 00000000 00000000 00000000 1c9c26b9 |
| 13 | [104] | 00000100 00000000 00000000 0000049d | [21] | 00000000 00000000 00000000 0026d9e3 |
| 14 | [112] | 00010000 00000000 00000000 0000014f | [14] | 00000000 00000000 00000000 00004377 |
| 15 | [120] | 01000000 00000000 00000000 000000e7 | [7] | 00000000 00000000 00000000 00000089 |
| 16 | [128] | 1 00000000 00000000 00000000 00000000 | [0] | 00000000 00000000 00000000 00000001 |

*Table 4.4: Polynomials used when creating $K'_c$. .* [1]

1. All polynomials are in hexadecimal notation. The LSB is in the rightmost position.

In Figure 4.5, all bits are shifted into the LFSRs, starting with the least significant bit (LSB). For instance, from the third octet of the address, BD_ADDR[2], first $BD\_ADDR_{16}$ is entered, followed by $BD\_ADDR_{17}$, etc. Furthermore, $CL_0$ corresponds to $CLK_1$,..., $CL_{25}$ corresponds to $CLK_{26}$.

Note that the output symbols $x_t^i$, $i = 1, \ldots, 4$ are taken from the positions 24, 24, 32, and 32 for $LFSR_1$, $LFSR_2$,$LFSR_3$, and $LFSR_4$, respectively (counting the leftmost position as number 1).

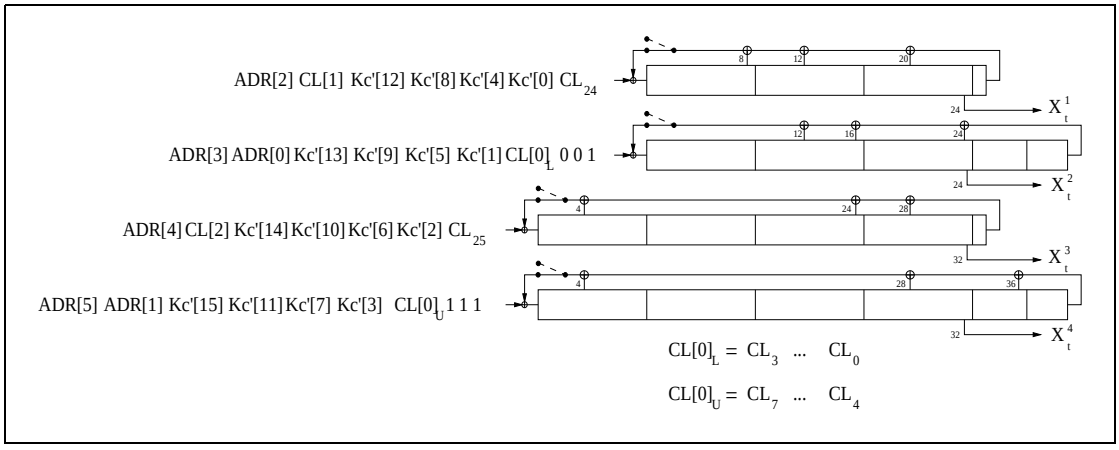



*Figure 4.5: Arranging the input to the LFSRs.*

In Figure 4.6, the 128 binary output symbols $Z_0$,..., $Z_{127}$ are arranged in octets denoted Z[0],..., Z[15]. The LSB of Z[0] corresponds to the first of these symbols, the MSB of Z[15] is the last output from the generator. These bits shall be loaded into the LFSRs according to the figure. It is a parallel load and no update of the blend registers is done. The first output symbol is generated at the same time. The octets shall be written into the registers with the LSB in the leftmost position (i.e. the opposite of before). For example, $Z_{24}$ is loaded into position 1 of $LFSR_4$.

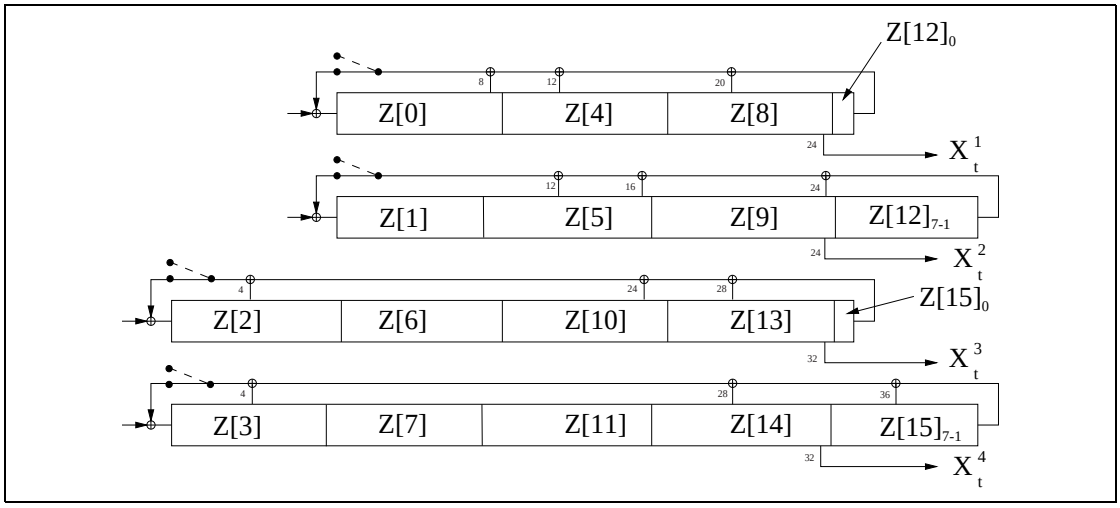



*Figure 4.6: Distribution of the 128 last generated output symbols within the LFSRs.*

## 4.6 KEY STREAM SEQUENCE

When the initialization is finished, the output from the summation combiner is used for encryption/decryption. The first bit to use shall be the one produced at the parallel load, i.e. at $t = 240$. The circuit shall be run for the entire length of the current payload. Then, before the reverse direction is started, the entire initialization process shall be repeated with updated values on the input parameters.

Sample data of the encryption output sequence can be found in [Part G] Section 1 on page 953. A necessary, but not sufficient, condition for all Bluetooth compliant implementations of encryption is to produce these encryption streams for identical initialization values.

# 5 AUTHENTICATION

Authentication uses a challenge-response scheme in which a claimant's knowledge of a secret key is checked through a 2-move protocol using symmetric secret keys. The latter implies that a correct claimant/verifier pair share the same secret key, for example K. In the challenge-response scheme the verifier challenges the claimant to authenticate a random input (the challenge), denoted by $AU\_RAND_A$, with an authentication code, denoted by $E_1$, and return the result SRES to the verifier, see Figure 5.1 on page 1082. This figure also shows that the input to $E_1$ consists of the tuple $AU\_RAND_A$ and the Bluetooth device address (BD_ADDR) of the claimant. The use of this address prevents a simple reflection attack[1]. The secret K shared by devices A and B is the current link key.

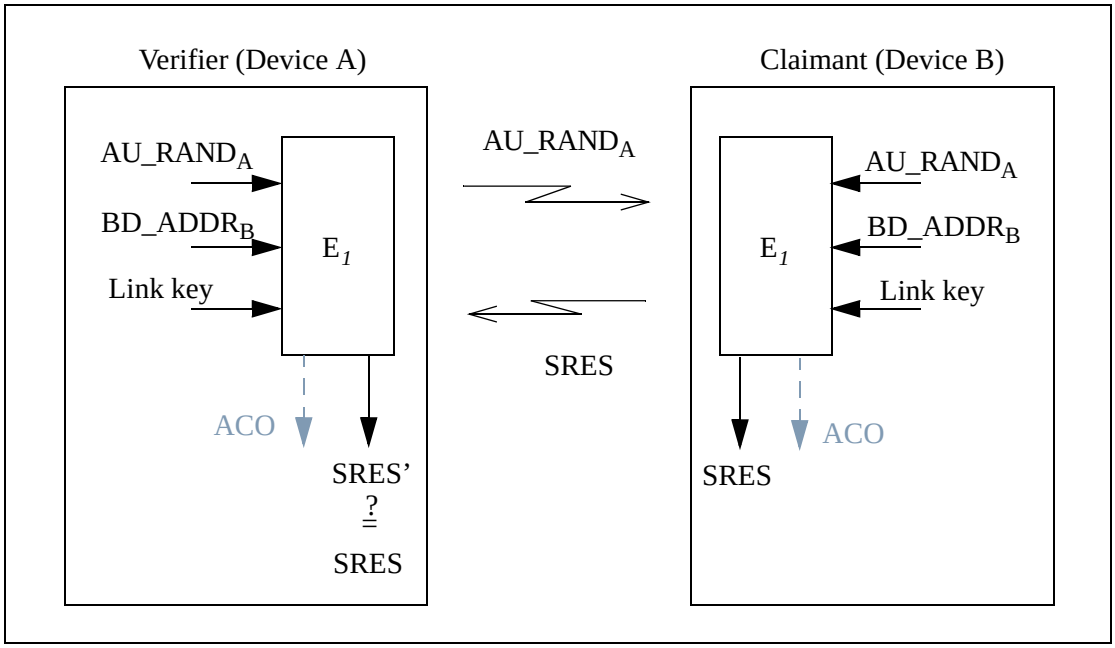



*Figure 5.1: Challenge-response for the Bluetooth.*

The challenge-response scheme for symmetric keys is depicted in Figure 5.2 on page 1083.

1. The reflection attack actually forms no threat because all service requests are dealt with on a FIFO bases. When preemption is introduced, this attack is potentially dangerous.

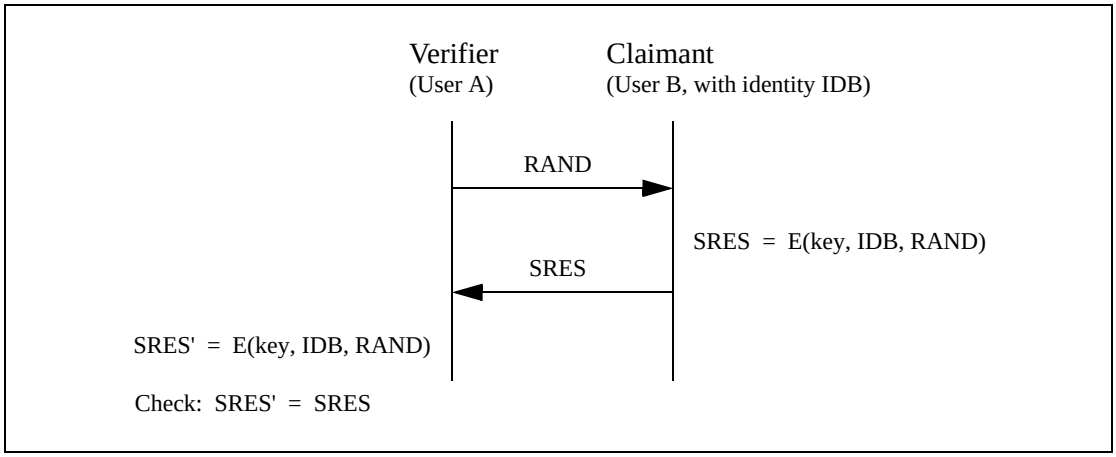



*Figure 5.2: Challenge-response for symmetric key systems.*

The verifier is not required to be the master. The application indicates which device has to be authenticated. Some applications only require a one-way authentication. However, some peer-to-peer communications, should use a mutual authentication in which each device is subsequently the challenger (verifier) in two authentication procedures. The LM shall process authentication preferences from the application to determine in which direction(s) the authentication(s) takes place. For mutual authentication with the devices of Figure 5.1 on page 1082, after device A has successfully authenticated device B, device B could authenticate device A by sending an $AU\_RAND_B$ (different from the $AU\_RAND_A$ that device A issued) to device A, and deriving the SRES and SRES' from the new $AU\_RAND_B$, the address of device A, and the link key $K_{AB}$.

If an authentication is successful the value of ACO as produced by $E_1$ shall be retained.

## 5.1 REPEATED ATTEMPTS

When the authentication attempt fails, a waiting interval shall pass before the verifier will initiate a new authentication attempt to the same claimant, or before it will respond to an authentication attempt initiated by a device claiming the same identity as the failed device. For each subsequent authentication failure, the waiting interval shall be increased exponentially. That is, after each failure, the waiting interval before a new attempt can be made, could be for example, twice as long as the waiting interval prior to the previous attempt[1]. The waiting interval shall be limited to a maximum.

The maximum waiting interval depends on the implementation. The waiting time shall exponentially decrease to a minimum when no new failed attempts are made during a certain time period. This procedure prevents an intruder from repeating the authentication procedure with a large number of different keys.

To protect a device's private key, a device should implement a method to prevent an attacker from retrieving useful information about the device's private key using invalid public keys. For this purpose, a device can use one of the following methods:

- Change its private key after three failed attempts from any BD_ADDR and after 10 successful pairings from any BD_ADDR; or after a combination of these such that 3 successful pairings count as one failed pairing; or
- Verify that the received public keys from any BD_ADDR are on the correct curve; or
- Implement elliptic curve point addition and doubling using formulas that are valid only on the correct curve.

1. Another appropriate value larger than 1 may be used.

# 6 THE AUTHENTICATION AND KEY-GENERATING FUNCTIONS

This section describes the algorithms used for authentication and key generation.

## 6.1 THE AUTHENTICATION FUNCTION $E_1$

The authentication function $E_1$ is a computationally secure authentication code$E_1$. uses the encryption function SAFER+. The algorithm is an enhanced version of an existing 64-bit block cipher SAFER-SK128, and it is freely available. In the following discussion, the block cipher will be denoted as the function $A_r$ which maps using a 128-bit key, a 128-bit input to a 128-bit output, i.e.

$$A_r: \{0,1\}^{128} \times \{0,1\}^{128} \rightarrow \{0,1\}^{128}$$
$$(k \times x) \mapsto t. \qquad \text{(EQ 11)}$$

The details of $A_r$ are given in the next section. The function $E_1$ is constructed using $A_r$ as follows

$$E_1: \{0,1\}^{128} \times \{0,1\}^{128} \times \{0,1\}^{48} \rightarrow \{0,1\}^{32} \times \{0,1\}^{96}$$
$$(K, RAND, address) \mapsto (SRES, ACO), \qquad \text{(EQ 12)}$$

where $SRES = Hash(K, RAND, address, 6)[0, \ldots, 3]$, where $Hash$ is a keyed hash function defined as[1],

$$Hash: \{0,1\}^{128} \times \{0,1\}^{128} \times \{0,1\}^{8 \times L} \times \{6,12\} \rightarrow \{0,1\}^{128}$$
$$(K, I_1, I_2, L) \mapsto A'_r([\tilde{K}], [E(I_2, L) +_{16} (A_r(K, I_1) \hat{\Uparrow}_{16} I_1)]), \qquad \text{(EQ 13)}$$

and where

$$E: \{0,1\}^{8 \times L} \times \{6,12\} \rightarrow \{0,1\}^{8 \times 16}$$
$$(X[0, \ldots, L-1], L) \mapsto (X[i(\text{mod } L)] \text{ for } i = 0...15), \qquad \text{(EQ 14)}$$

is an expansion of the $L$ octet word $X$ into a 128-bit word. The function $A_r$ is evaluated twice for each evaluation of $E_1$. The key $\tilde{K}$ for the second use of $A_r$ (actually $A'_r$) is offset from $K$ as follows[2]

1. The operator $+_{16}$ denotes bytewise addition mod 256 of the 16 octets, and the operator $\oplus_{16}$ denotes bytewise XORing of the 16 octets.
2. The constants are the largest primes below 257 for which 10 is a primitive root.

$$
\begin{aligned}
\tilde{K}[0] &= (K[0] + 233) \mod 256, & \tilde{K}[1] &= K[1] \oplus 229, \\
\tilde{K}[2] &= (K[2] + 223) \mod 256, & \tilde{K}[3] &= K[3] \oplus 193, \\
\tilde{K}[4] &= (K[4] + 179) \mod 256, & \tilde{K}[5] &= K[5] \oplus 167, \\
\tilde{K}[6] &= (K[6] + 149) \mod 256, & \tilde{K}[7] &= K[7] \oplus 131, \\
\tilde{K}\{8\} &= K[8] \oplus 233, & \tilde{K}[9] &= (K[9] + 229) \mod 256, \\
\tilde{K}[10] &= K[10] \oplus 223, & \tilde{K}[11] &= (K[11] + 193) \mod 256, \\
\tilde{K}[12] &= K[12] \oplus 179, & \tilde{K}[13] &= (K[13] + 167) \mod 256, \\
\tilde{K}[14] &= K[14] \oplus 149, & \tilde{K}[15] &= (K[15] + 131) \mod 256.
\end{aligned}
\qquad \text{(EQ 15)}
$$

A data flowchart of the computation of $E_1$ is shown in Figure 6.1 on page 1086 $E_1$. is also used to deliver the parameter ACO (Authenticated Ciphering Offset) that is used in the generation of the ciphering key by $E_3$, see equations (EQ 3) on page 1068 and (EQ 23) on page 1091. The value of ACO is formed by the octets 4 through 15 of the output of the hash function defined in (EQ 13) on page 1085:

$$\text{ACO} = \text{Hash}(K, \text{RAND}, \text{address}, 6)[4, \ldots, 15]. \qquad \text{(EQ 16)}$$

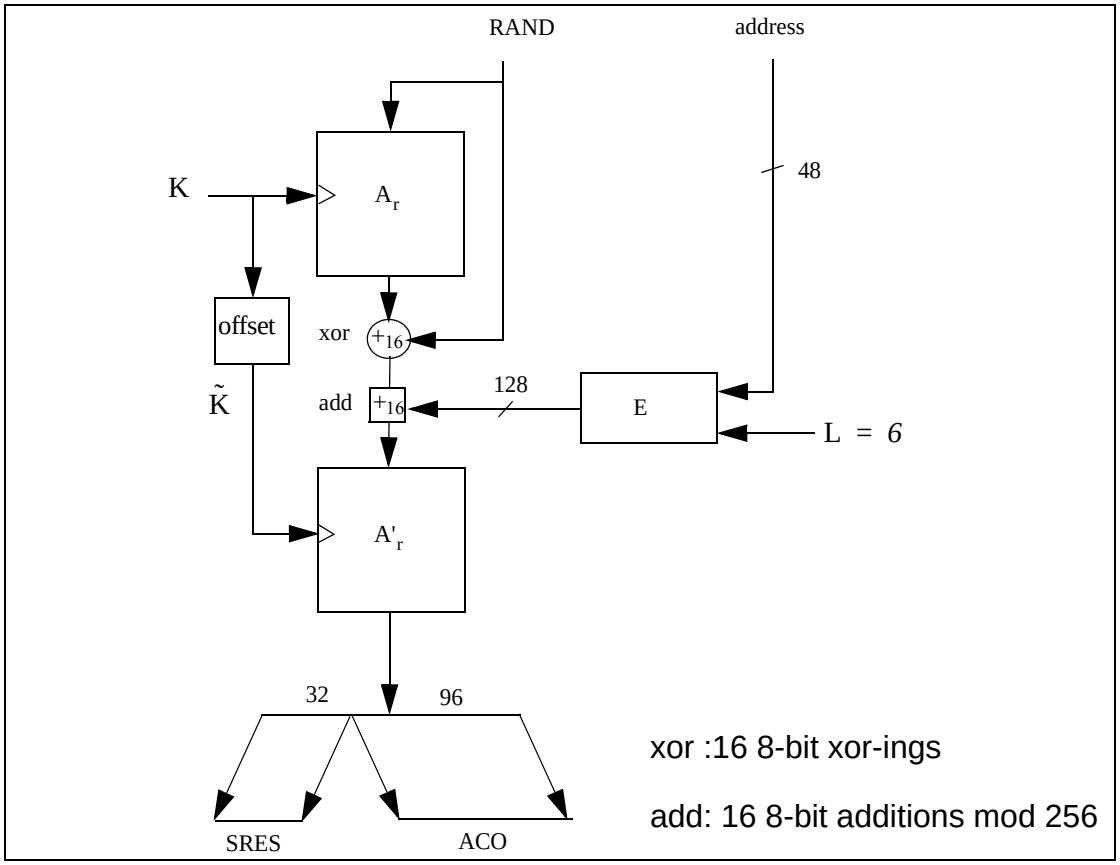



*Figure 6.1: Flow of data for the computation of $E_1$.*

## 6.2 THE FUNCTIONS $A_r$ AND $A'_r$

The function $A_r$ is identical to SAFER+. It consists of a set of 8 layers, (each layer is called a round) and a parallel mechanism for generating the sub keys $K_p[j]$, $p = 1, 2, \ldots, 17$, which are the round keys to be used in each round. The function will produce a 128-bit result from a 128-bit random input string and a 128-bit key. Besides the function $A_r$, a slightly modified version referred to as $A'_r$ is used in which the input of round 1 is added to the input of round 3. This is done to make the modified version non-invertible and prevents the use of $A'_r$ (especially in $E_{2x}$) as an encryption function. See Figure 6.2 on page 1088 for details.

### 6.2.1 The round computations

The computations in each round are a composition of encryption with a round key, substitution, encryption with the next round key, and, finally, a Pseudo Hadamard Transform (PHT). The computations in a round shall be as shown in Figure 6.2 on page 1088. The sub keys for round $r, r = 1, 2, \ldots, 8$ are denoted $K_{2r-1}[j]$, $K_{2r}[j]$, $j = 0, 1, \ldots, 15$. After the last round $K_{17}[j]$ is applied identically to all previous odd numbered keys.

### 6.2.2 The substitution boxes “e” and “l”

In Figure 6.2 on page 1088 two boxes are shown, marked “e” and “l”. These boxes implement the same substitutions as are used in SAFER+; i.e. they implement

$$\begin{aligned} e, l &: \{0, \ldots, 255\} \rightarrow \{0, \ldots, 255\}, \\ e &: i \mapsto (45^i \pmod{257}) \pmod{256}, \\ l &: i \mapsto j \text{ s.t. } i = e(j). \end{aligned}$$

Their role, as in the SAFER+ algorithm, is to introduce non-linearity.

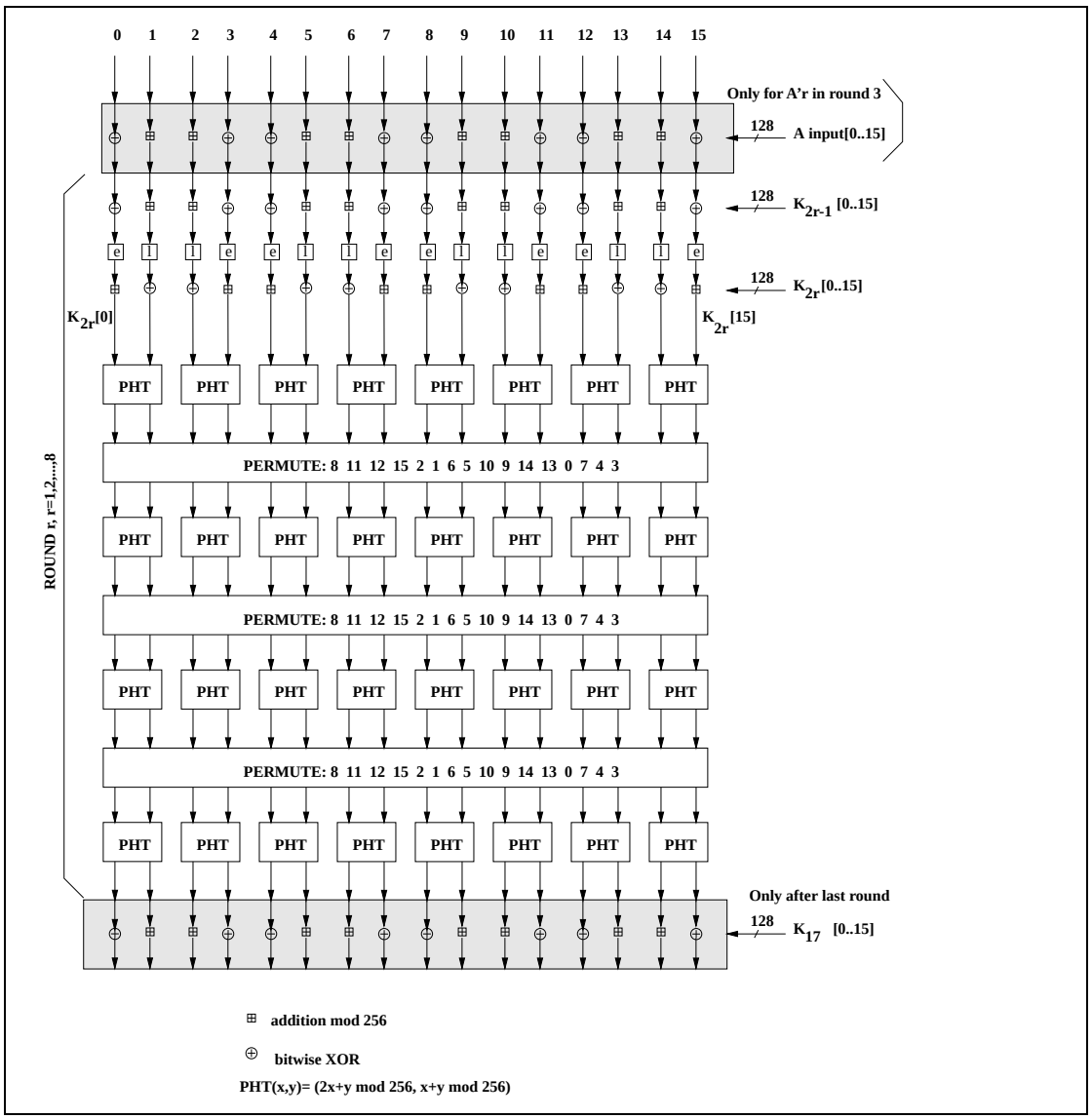



*Figure 6.2: One round in $A_r$ and $A'_r$.*

In Section 6.2, the permutation boxes show how input byte indices are mapped onto output byte indices. Thus, position 8 is mapped on position 0 (leftmost), position 11 is mapped on position 1, etc.

### 6.2.3 Key scheduling

In each round, 2 batches of 16 octet-wide keys are needed. These round keys are derived as specified by the key scheduling in SAFER+. Figure 6.3 on page 1089 gives an overview of how the round keys $K_p[j]$ are determined. The bias vectors $B_2$, $B_3$, ..., $B_{17}$ shall be computed according to following equation:

$$B_p[i] = ((45^{(45^{17p+i+1} \bmod 257)} \bmod 257) \bmod 256), \text{ for } i = 0, \ldots, 15. \qquad \text{(EQ 17)}$$

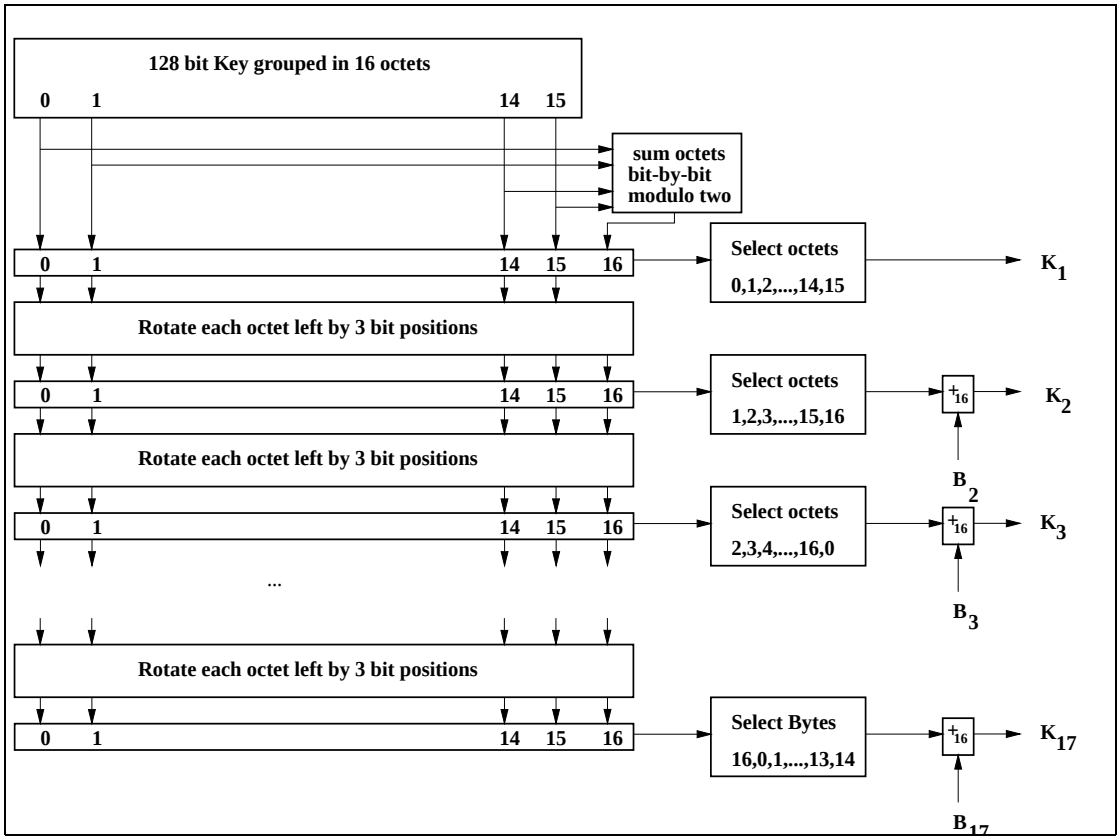



*Figure 6.3: Key scheduling in $A_r$.*

## 6.3 $E_2$-KEY GENERATION FUNCTION FOR AUTHENTICATION

The key used for authentication shall be derived through the procedure that is shown in Figure 6.4 on page 1091. The figure shows two modes of operation for the algorithm. In the first mode, $E_{21}$ produces a 128-bit kink key, $K$, using a 128-bit RAND value and a 48-bit address. This mode shall be utilized when creating unit keys and combination keys. In the second mode, $E_{22}$ produces a 128-bit link key, $K$, using a 128-bit RAND value and an $L$ octet user PIN. The second mode shall be used to create the initialization key, and also when a master key is to be generated.

When the initialization key is generated, the PIN is augmented with the BD_ADDR, see Section 3.2.1 on page 1066 for which address to use. The augmentation shall always start with the least significant octet of the address immediately following the most significant octet of the PIN. Since the maximum length of the PIN used in the algorithm cannot exceed 16 octets, it is possible that not all octets of BD_ADDR will be used.

This key generating algorithm again exploits the cryptographic function $E_2$. for mode 1 (denoted $E_{21}$) is computed according to following equations:

$$\begin{aligned} E_{21}:\ \{0,1\}^{128} \times \{0,1\}^{48} &\rightarrow \{0,1\}^{128} \\ (\text{RAND, address}) &\mapsto A'_r(X, Y) \end{aligned} \qquad \text{(EQ 18)}$$

where (for mode 1)

$$\begin{cases} X = \text{RAND}[0 \ldots 14] \cup (\text{RAND}[15] \oplus 6) \\ Y = \bigcup_{i=0}^{15} \text{address}[i \pmod 6] \end{cases} \qquad \text{(EQ 19)}$$

Let $L$ be the number of octets in the user PIN. The augmenting is defined by

$$\text{PIN}' = \begin{cases} \text{PIN}[0 \ldots L-1] \cup \text{BD\_ADDR}[0 \ldots \min\{5, 15-L\}], & L < 16, \\ \text{PIN}[0 \ldots L-1], & L = 16, \end{cases} \qquad \text{(EQ 20)}$$

Then, in mode 2, $E_2$ (denoted $E_{22}$) is

$$\begin{aligned} E_{22}:\ \{0,1\}^{8L'} \times \{0,1\}^{128} \times \{1, 2, \ldots, 16\} &\rightarrow \{0,1\}^{128} \\ (\text{PIN', RAND, } L') &\mapsto A'_r(X, Y) \end{aligned} \qquad \text{(EQ 21)}$$

where

$$\begin{cases} X = \bigcup_{i=0}^{15} \text{PIN}'[i \pmod{L'}], \\ Y = \text{RAND}[0 \ldots 14] \cup (\text{RAND}[15] \oplus L'), \end{cases} \qquad \text{(EQ 22)}$$

and $L' = \min\{16, L + 6\}$ is the number of octets in PIN'.

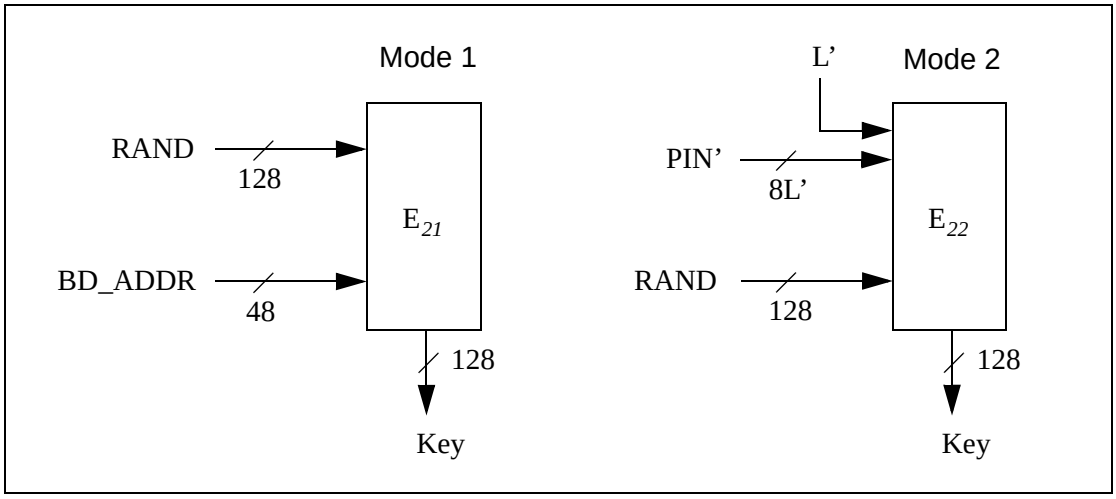



*Figure 6.4: Key generating algorithm $E_2$ and its two modes. Mode 1 is used for unit and combination keys, while mode 2 is used for $K_{init}$ and $K_{master}$.*

## 6.4 $E_3$-KEY GENERATION FUNCTION FOR ENCRYPTION

The ciphering key $K_C$ used by $E_0$ shall be generated by $E_3$. The function $E_3$ is constructed using $A'_r$ as follows

$$E_3: \{0,1\}^{128} \times \{0,1\}^{128} \times \{0,1\}^{96} \rightarrow \{0,1\}^{128}$$
$$(K, RAND, COF) \mapsto Hash(K, RAND, COF, 12) \qquad \text{(EQ 23)}$$

where $Hash$ is the hash function as defined by (EQ 13) on page 1085. The key length produced is 128 bits. However, before use within $E_0$, the encryption key $K_C$ is shortened to the correct encryption key length, as described in Section 4.5 on page 1078. A block scheme of $E_3$ is depicted in Figure 6.5.

The value of COF is determined as specified by equation (EQ 3) on page 1068.

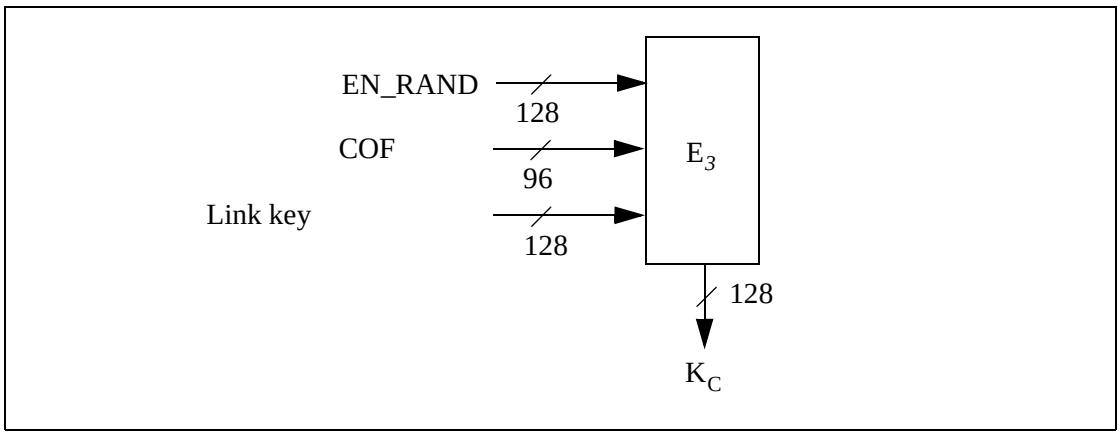



*Figure 6.5: Generation of the encryption key*

# 7 SECURE SIMPLE PAIRING

The Secure Simple Pairing security functions and procedures are described in this section. In addition, a cryptographic analysis of each procedure is provided.

There are five phases of Secure Simple Pairing:

- Phase 1: Public key exchange
- Phase 2: Authentication Stage 1
- Phase 3: Authentication Stage 2
- Phase 4: Link key calculation
- Phase 5: LMP Authentication and Encryption

Phases 1, 3, 4 and 5 are the same for all protocols whereas phase 2 (Authentication Stage 1) is different depending on the protocol used. Distributed through these five phases are 13 steps.

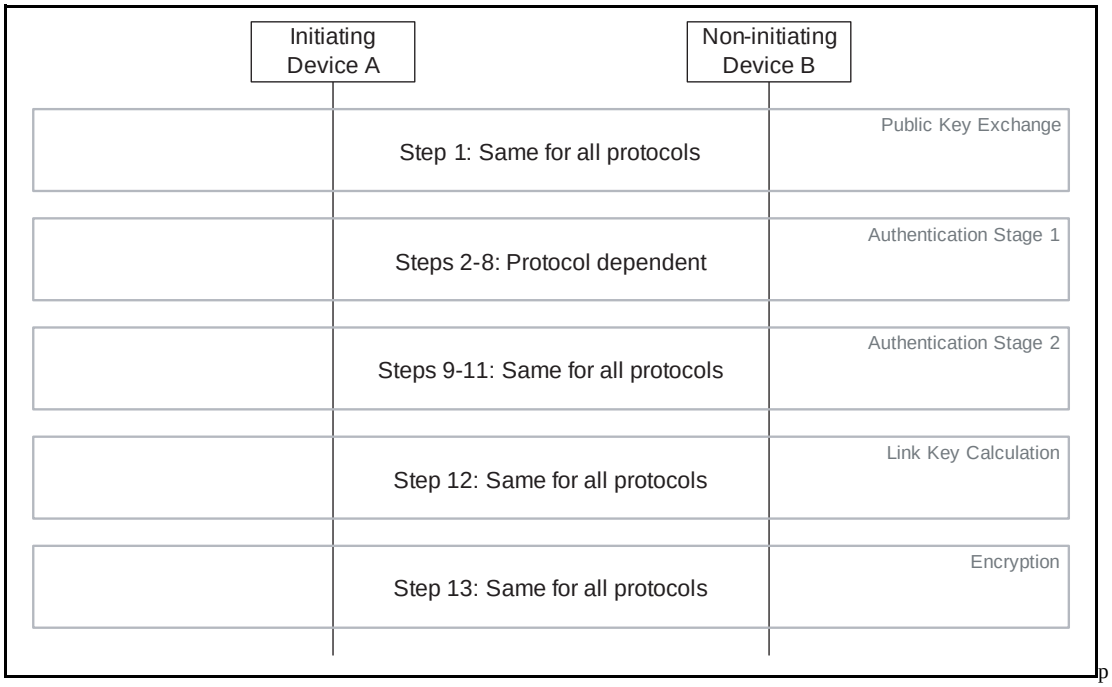



*Figure 7.1: Simple Pairing Security Phases*

The terminology used in the security sections is defined in Table 7.1:

| Term | Definition |
|---|---|
| Cx | Commitment value from device X |
| Cxi | $i^{th}$ commitment value from device X. Only used in the passkey entry protocol |

*Table 7.1: Terminology*

| Term | Definition |
|---|---|
| DHKey | Diffie Hellman key |
| Ex | Check value from device X |
| f1( ) | Used to generate the 128-bit commitment values Ca and Cb |
| f2( ) | Used to compute the link key and possible other keys from the DHKey and random nonces |
| f3( ) | Used to compute check values Ea and Eb in Authentication Stage 2 |
| g( ) | Used to compute numeric check values |
| h2( ) | Used to compute Generic AMP and Dedicated AMP keys |
| IOcapA | IO capabilities of device A |
| IOcapB | IO capabilities of device B |
| LK | Link Key |
| Nx | Nonce (unique random value) from device X |
| Nxi | $i^{th}$ nonce (unique random value) from device X. Only used in the passkey entry protocol |
| PKx | Public Key of device X |
| rx | Random value generated by device X |
| rxi | Bit i of the random value rx. Only used in the passkey entry protocol |
| SKx | Secret (Private) Key of device X |
| Vx | Confirmation value on device X. Only used in the numeric compare protocol. |
| X | BD_ADDR of device X |

*Table 7.1: Terminology*

## 7.1 PHASE 1: PUBLIC KEY EXCHANGE

Initially, each device generates its own Elliptic Curve Diffie-Hellman (ECDH) public-private key pair (step 1). This key pair needs to be generated only once per device and may be computed in advance of pairing. A device may, at any time, choose to discard its public-private key pair and generate a new one, although there is not a requirement to do so.

Pairing is initiated by the initiating device sending its public key to the receiving device (step 1a). The responding device replies with its own public key (step 1b) These public keys are not regarded as secret although they may identify the devices. Note that steps 1a and 1b are the same in all three protocols.

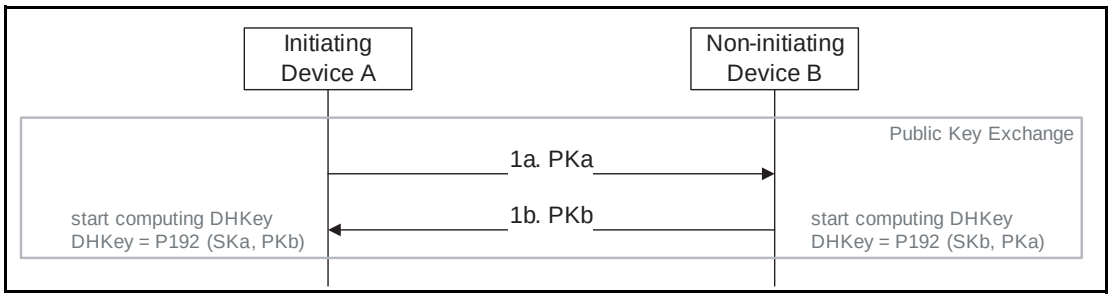



*Figure 7.2: Public Key Exchange Details*

## 7.2 PHASE 2: AUTHENTICATION STAGE 1

Authentication Stage 1 has three different protocols: Numeric Comparison, Out-of-Band, and Passkey Entry. Note that there is not a separate protocol for the Just Works association model. This association model shares the Numeric Comparison protocol.

The protocol is chosen based on the IO capabilities of the two devices.

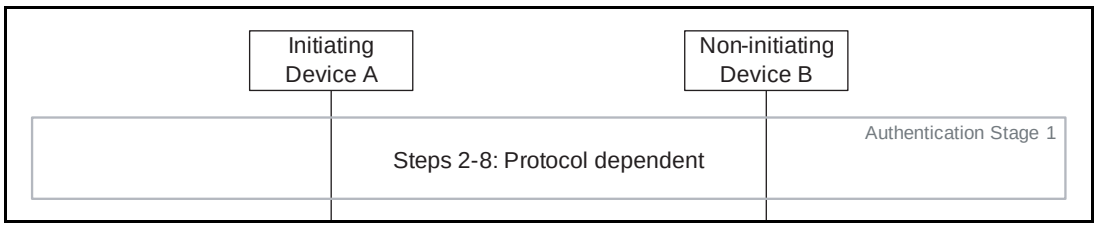



*Figure 7.3: Authentication Stage 1 (High Level)*

The three protocols are described in the following sections.

### 7.2.1 Authentication Stage 1: Numeric Comparison Protocol

The Numeric Comparison protocol provides limited protection against active "man-in-the-middle" (MITM) attacks as an active man-in-the-middle will succeed with a probability of 0.000001 on each iteration of the protocol. Provided that there is no MITM at the time the pairing is performed, the shared Link Key that is generated is computationally secure from even a passive eavesdropper that may have been present during the pairing.

The sequence diagram of Authentication Stage 1 for the Numeric Comparison protocol from the cryptographic point of view is shown below:

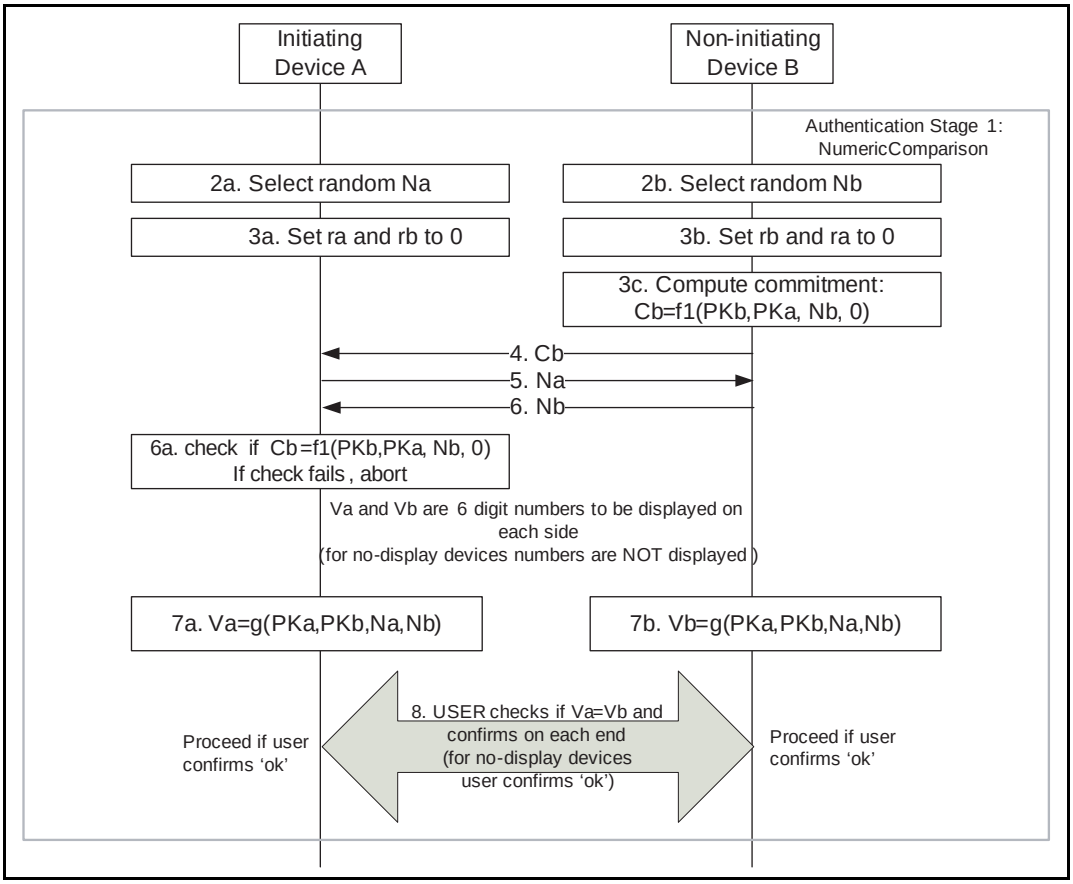



*Figure 7.4: Authentication Stage 1: Numeric Comparison Protocol Details*

After the public keys have been exchanged, each device selects a pseudo-random 128-bit nonce (step 2). This value is used to prevent replay attacks and must be freshly generated with each instantiation of the pairing protocol. This value should be generated directly from a physical source of randomness or with a good pseudo-random generator seeded with a random value from a physical source.

Following this the responding device then computes a commitment to the two public keys that have been exchanged and its own nonce value (step 3c). This commitment is computed as a one-way function of these values and is transmitted to the initiating device (step 4). The commitment prevents an attacker from changing these values at a later time.

The initiating and responding devices then exchange their respective nonce values (steps 5 and 6) and the initiating device confirms the commitment (step 6a). A failure at this point indicates the presence of an attacker or other transmission error and causes the protocol to abort. The protocol may be repeated with or without the generation of new public-private key pairs, but new nonces must be generated if the protocol is repeated.

Assuming that the commitment check succeeds, the two devices each compute 6-digit confirmation values that are displayed to the user on their respective devices (steps 7a, 7b, and 8). The user is expected to check that these 6-

digit values match and to confirm if there is a match. If there is no match, the protocol aborts and, as before, new nonces must be generated if the protocol is to be repeated.

An active MITM must inject its own key material into this process to have any effect other than denial-of-service. A simple MITM attack will result in the two 6-digit display values being different with probability 0.999999. A more sophisticated attack may attempt to engineer the display values to match, but this is thwarted by the commitment sequence. If the attacker first exchanges nonces with the responding device, it must commit to the nonce that it will use with the initiating device before it sees the nonce from the initiating device. If the attacker first exchanges nonces with the initiating device, it must send a nonce to the responding device before seeing the nonce from the responding device. In each case, the attacker must commit to at least the second of its nonces before knowing the second nonce from the legitimate devices. It therefore cannot choose its own nonces in such a way as to cause the display values to match.

### 7.2.2 Authentication Stage 1: Out of Band Protocol

The Out-of-Band protocol is used when authentication information has been received by at least one of the devices and indicated in the OOB Authentication Data Present parameter in the LMP IO capability exchange sequence. The mode in which the discovery of the peer device is first done in-band and then followed by the transmission of authentication parameters through OOB interface is not supported. The sequence diagram for Authentication Stage 1 for Out of Band from the cryptographic point of view is shown below:

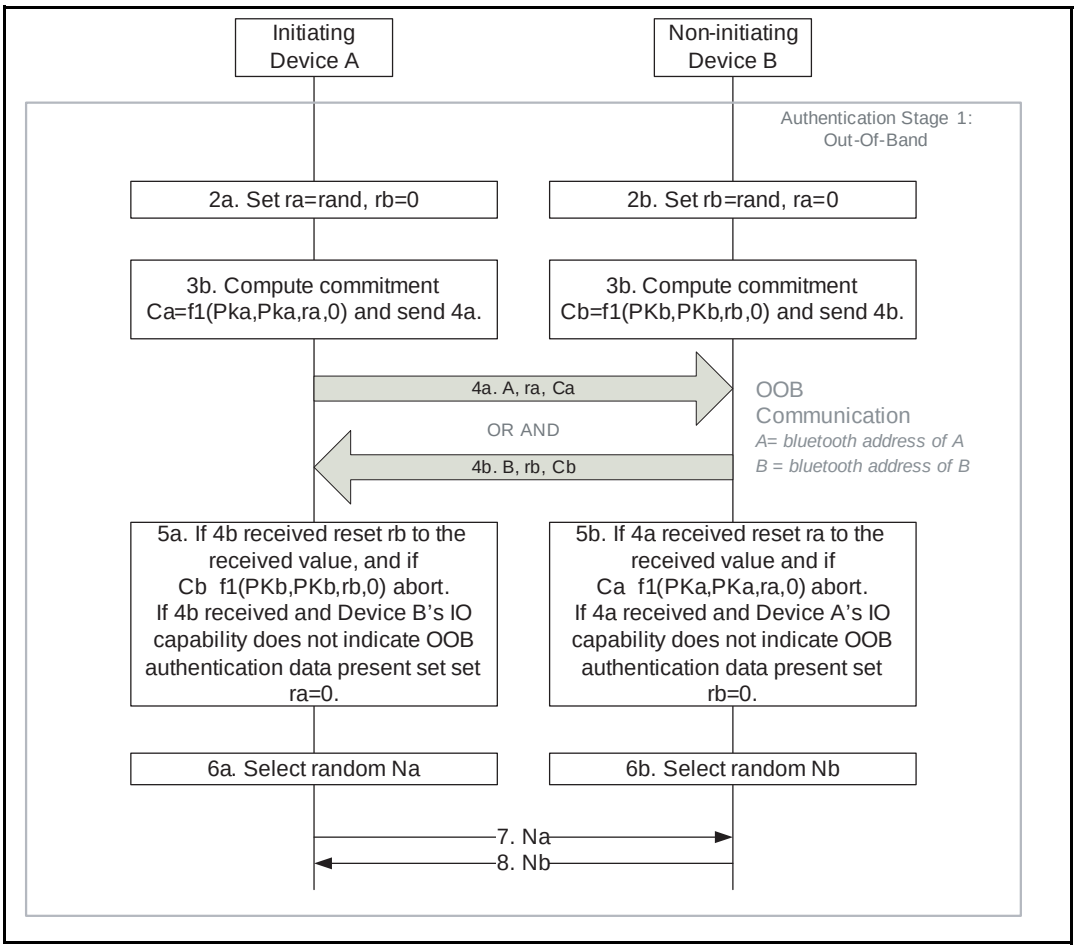



*Figure 7.5: Authentication Stage 1: Out of Band Details*

**Principle of operation**: If both devices can transmit and/or receive data over an out-of-band channel, then mutual authentication will be based on the commitments of the public keys (Ca and Cb) exchanged OOB in Authentication stage 1. If OOB communication is possible only in one direction (e.g., one device can only be read as e.g. is the case of a device equipped with a passive NFC tag), then authentication of the device receiving the OOB communication will be based on that device knowing a random number r sent via OOB. In this case, r must be secret: r can be created afresh every time, or access to the device sending r must be restricted. If r is not sent by a device, it is assumed to be 0 by the device receiving the OOB information in step 4a or 4b.

If the OOB communication cannot be tampered with (i.e. immune to a MITM attack), this association method is also not susceptible to MITM attack. Also, in the Out of Band protocol the size of authentication parameters (Ca, Cb and ra, rb) need not be restricted by what the user can comfortably read or type. For that reason the Out of Band protocol can be more secure than using the Numeric Comparison or Passkey Entry protocols. However, both devices need to have matching OOB interfaces.

**Roles of A and B**: The OOB Authentication Stage 1 protocol is symmetric with respect to the roles of A and B. It does not require that device A always will ini-

tiate pairing and it automatically resolves asymmetry in the OOB communication, e.g. in the case when one of the devices has an NFC tag and can only transmit OOB.

**Order of steps**: The public key exchange must happen before the verification step 5. In the diagram the in-band public key exchange between the devices (step 1) is done before the OOB communication (step 4). But when the pairing is initiated by an OOB interface, public key exchange will happen after the OOB communication (step 1 will be between steps 4 and 5).

Values of ra and rb: Since the direction of the peer's OOB interface cannot be verified before the OOB communication takes place, a device should always generate and if possible transmit through its OOB interface a random number r to the peer. Each device applies the following rules locally to set the values of its own r and the value of the peer's r:

1. Initially, r of the device is set to a random number and r of the peer is set to 0 (step 2).
2. If a device has received OOB, it sets the peer's r value to what was sent by the peer (Step 5).
3. If the remote device's OOB Authentication Data parameter sent in the LMP IO capabilities exchange sequence is set to No OOB Data Received, it sets its own r value to 0 (Step 5)

These rules ensure that when entering Authentication Stage 2, both A and B have the same values for ra and rb if OOB communication took place.

### 7.2.3 Authentication Stage 1: Passkey Entry Protocol

The Passkey Entry protocol is used when LMP IO capability exchange sequence indicates that Passkey Entry shall be used.

The sequence diagram for Authentication Stage 1 for Passkey Entry from the cryptographic point of view is shown in Figure 7.6.

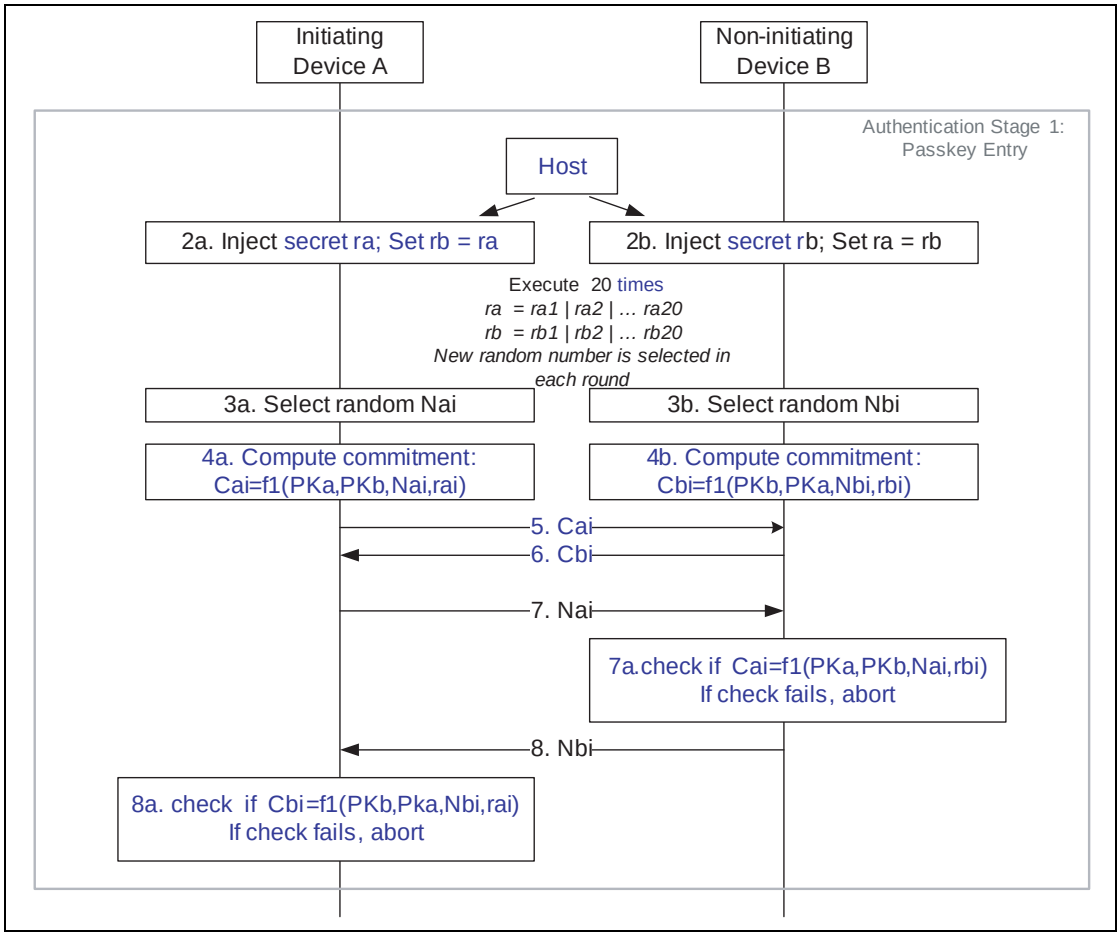



*Figure 7.6: Authentication Stage 1: Passkey Entry Details*

The user inputs an identical Passkey into both devices. Alternately, the Passkey may be generated and displayed on one device, and the user then inputs it into the other (step 2). This short shared key will be the basis of the mutual authentication of the devices. Steps 3 through 8 are repeated k times for a k-bit Passkey --e.g., k=20 for a 6-digit Passkey (999999=0xF423F).

In Steps 3-8, each side commits to each bit of the Passkey, using a long nonce (128 bits), and sending the hash of the nonce, the bit of the Passkey, and both public keys to the other party. The parties then take turns revealing their commitments until the entire Passkey has been mutually disclosed. The first party to reveal a commitment for a given bit of the Passkey effectively reveals that bit of the Passkey in the process, but the other party then has to reveal the corresponding commitment to show the same bit value for that bit of the Passkey, or else the first party will then abort the protocol, after which no more bits of the Passkey are revealed.

This "gradual disclosure" prevents leakage of more than 1 bit of un-guessed Passkey information in the case of a MITM attack. A MITM attacker with only partial knowledge of the Passkey will only receive one incorrectly-guessed bit of the Passkey before the protocol fails. Hence, a MItM attacker who engages

first one side, then the other will only gain an advantage of at most two bits over a simple brute-force guesser which succeeds with probability 0.000001.

The long nonce is included in the commitment hash to make it difficult to brute-force even after the protocol has failed. The public Diffie-Hellman values are included to tie the Passkey protocol to the original ECDH key exchange, to prevent a MItM from substituting the attacker's public key on both sides of the ECDH exchange in standard MItM fashion.

At the end of this stage, Na is set to $Na_{20}$ and Nb is set to $Nb_{20}$ for use in Authentication Stage 2.

## 7.3 PHASE 3: AUTHENTICATION STAGE 2

The second stage of authentication then confirms that both devices have successfully completed the exchange. This stage is identical in all three protocols.

Each device computes a new confirmation value that includes the previously exchanged values and the newly derived shared key (step 9). The initiating device then transmits its confirmation value which is checked by the responding device (step 10). If this check fails, it indicates that the initiating device has not confirmed the pairing, and the protocol must be aborted. The responding device then transmits its confirmation value which is checked by the initiating device (step 11). A failure indicates that the responding device has not confirmed the pairing and the protocol should abort.

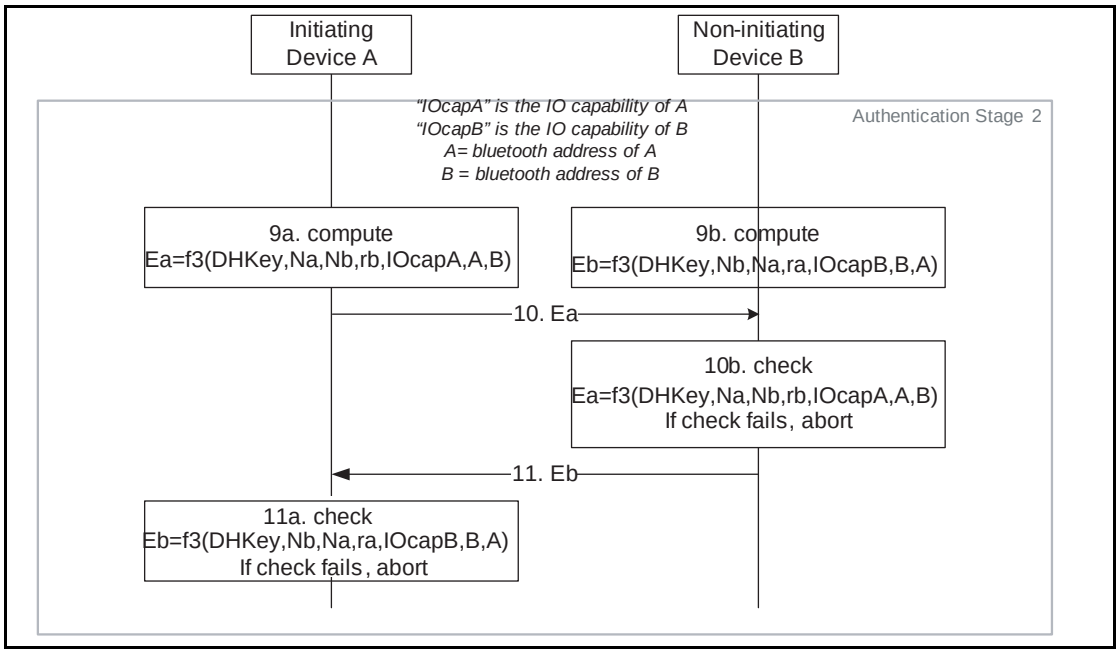



*Figure 7.7: Authentication Stage 2*

## 7.4 PHASE 4: LINK KEY CALCULATION

Once both sides have confirmed the pairing, a link key is computed from the derived shared key and the publicly exchanged data (step 12). The nonces ensure the freshness of the key even if long-term ECDH values are used by both sides. This link key is used to maintain the pairing.

When computing the link key both parties shall input the parameters in the same order to ensure that both devices compute the same key: device A's parameters are those of the piconet master and device B's parameters are those of the piconet slave.

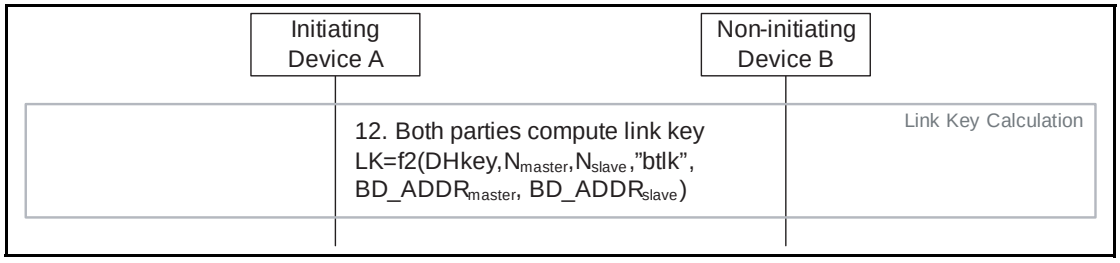



*Figure 7.8: Link Key Calculation*

## 7.5 PHASE 5: LMP AUTHENTICATION AND ENCRYPTION

The final phase in Simple Pairing is generating the encryption keys. This is the same as the final steps in legacy pairing.

## 7.6 ELLIPTIC CURVE DEFINITION

Simple Pairing uses the FIPS P-192 curve defined in [13]. Elliptic curves are specified by p, a, b and are of the form

$$E: y^2 = x^3 + ax + b \pmod p$$

For each value of b a unique curve can be developed. In NIST P-192:

$$a = \text{mod}(-3, p)$$

b is defined and its method of generation can be verified by using SHA-1 (with a given seed s and using $b^2c = -27 \pmod p$))

The following parameters are given:

- The prime modulus p, order r, base point x-coordinate Gx, base point y-coordinate Gy.
- The integers p and r are given in decimal form; bit strings and field elements are given in hex.

```
p = 6277101735386680763835789423207666416083908700390324961279
r = 6277101735386680763835789423176059013767194773182842284081
b = 64210519 e59c80e7 0fa7e9ab 72243049 feb8deec c146b9b1
```

```
Gx = 188da80e b03090f6 7cbf20eb 43a18800 f4ff0afd 82ff1012
Gy = 07192b95 ffc8da78 631011ed 6b24cdd5 73f977a1 1e794811
```

The function P192 is defined as follows. Given an integer u, 0 < u < r, and a point V on the curve E, the value P192(u,V) is computed as the x-coordinate of the $u^{th}$ multiple uV of the point V.

The private keys shall be between 1 and r/2, where r is the Order of the Abelian Group on the elliptic curve (e.g. between 1 and $2^{192}/2$).

## 7.7 CRYPTOGRAPHIC FUNCTION DEFINITIONS

In addition to computing the Elliptic Curve Diffie Hellman key, the Numeric Comparison, Out-of-Band and Passkey Entry protocols require four cryptographic functions. These functions are known as *f1*, *g*, *f2* and *f3*.

*f1* is used to generate the 128-bit commitment values Ca and Cb

*g* is used to compute the 6-digit numeric check values

*f2* is used to compute the link key and possible other keys from the DHKey and random nonces

*f3* is used to compute check values Ea and Eb in Authentication Stage 2.

The basic building block for these functions is based on as it already exists in the key derivation function E3 (which is the same as E1 but with a larger input).

Inside the f1, g, f2, and f3 cryptographic functions, when a multi-octet integer input parameter is used as input to the SHA-256 and HMAC-SHA-256 functions, the most significant octet of the integer parameter shall be the first octet of the stream and the least significant octet of the integer parameter shall be the last octet of the stream. The output of the f1, g, f2 and f3 cryptographic functions is a multi-octet integer where the first octet out of SHA-256 and HMAC-SHA-256 shall be the MSB and the last octet shall be the LSB of that parameter.

Inside the h2 cryptographic function, when a multi-octet integer input parameter is used as input to the SHA-256 and HMAC-SHA-256 functions, the most significant octet of the integer parameter shall be the last octet of the stream and the least significant octet of the integer parameter shall be the first octet of the stream. The output of the h2 cryptographic function is a multi-octet integer where the first octet out of SHA-256 and HMAC-SHA-256 shall be the LSB and the last octet shall be the MSB of that parameter.

### 7.7.1 The Simple Pairing Commitment Function f1

The commitments are computed with function f1. The definition of the Simple Pairing commitment function makes use of the MAC function HMAC based on $SHA\text{-}256_X$, which is denoted as $HMAC\text{-}SHA\text{-}256_X$ with 128-bit key X.

The inputs to the Simple Pairing function f1 are:

U is 192 bits

V is 192 bits

X is 128 bits

Z is 8 bits

Z is zero (i.e. 8 bits of zeros) for Numeric Comparison and OOB protocol. In the Passkey protocol the most significant bit of Z is set equal to one and the least significant bit is made up from one bit of the passkey e.g. if passkey is '1' then Z = 0x81 and if the passkey is '0' then Z = 0x80.

The output of the Simple Pairing f1 function is:

$$f1(U, V, X, Z) = \text{HMAC-SHA-256}_X (U \,||\, V \,||\, Z) / 2^{128}$$

The inputs to *f1* are different depending on the different protocols.

| Numeric Comparison | Out-Of-Band | Passkey Entry |
|---|---|---|
| Ca = f1(PKax, PKbx, Na, 0) | Ca = f1(PKax, PKax, Ra, 0) | Cai = f1(PKax, PKbx, Nai, rai) |
| Cb = f1(PKbx, PKax, Nb, 0) | Cb = f1(PKbx, PKbx, Rb, 0) | Cbi = f1(PKbx, PKax, Nbi, rbi) |

*Table 7.2: Inputs to f1 for the different protocols*

where PKax denotes the x-coordinate of the public key PKa of A. Similarly, PKbx denotes the x-coordinate of the public key PKb of B. Nai is the nonce value of $i^{th}$ round. For each round Nai value is a new 128 bit number. Similarly, rai is one bit value of the passkey expanded to 8 bits (either 0x80 or 0x81).

### 7.7.2 The Simple Pairing Numeric Verification Function *g*

The Simple Pairing *g* function is defined as follows:

The inputs to the Simple Pairing function *g* are:

U is 192 bits

V is 192 bits

X is 128 bits

Y is 128 bits

The output of the Simple Pairing g function is:

$$g(U, V, X, Y) = \text{SHA-256}(U \,||\, V \,||\, X \,||\, Y) \bmod 2^{32}$$

The numeric verification value is taken as six least significant digits of the 32-bit integer *g*(PKax, PKbx, Na, Nb) where PKax denotes the x-coordinate of the

public key PKa of A and PKbx denotes the x-coordinate of the public key PKb of B.

Output of SHA-256 is truncated to 32 bits by taking the least significant 32 bits of the output of SHA-256. This value is converted to decimal numeric value. The checksum used for numeric comparison is the least significant six digits.

$$\text{Compare Value} = g(U, V, X, Y) \bmod 10^6$$

For example, if output = 0x 01 2e b7 2a then decimal value = 19838762 and the checksum used for numeric comparison is 838762.

### 7.7.3 The Simple Pairing Key Derivation Function f2

The definition of the Simple Pairing key derivation function makes use of the MAC function HMAC based on SHA-256, which is denoted as HMAC-SHA-$256_W$ with 192-bit key W.

The inputs to the Simple Pairing function *f2* are:

W is 192 bits

$N_1$ is 128 bits

$N_2$ is 128 bits

keyID is 32 bits

$A_1$ is 48 bits

$A_2$ is 48 bits

The string "btlk" is mapped into keyID using extended ASCII[1] as follows:

keyID[0] = `0110 1011 (lsb)`

keyID[1] = `0110 1100`

keyID[2] = `0111 0100`

keyID[3] = `0110 0010`

KeyID = `0x62746c6b`

The output of the Simple Pairing f2 function is:

$$f2(W, N_1, N_2, \text{KeyID}, A_1, A_2) = \text{HMAC-SHA-256}_W (N1 \,||\, N2 \,||\, \text{KeyID} \,||\, A_1 \,||\, A_2) / 2^{128}$$

The output of *f2* is taken as the 128 most significant (leftmost) bits of the output of HMAC-SHA-256.

1. ISO/IEC 8859-1
(see www.iso.org)

The link key is then calculated as:

LK = f2(DHKey, N_master, N_slave, "btlk", BD_ADDR_master, BD_ADDR_slave)

### 7.7.4 The Simple Pairing Check Function f3

The definition of the Simple Pairing f3 check function makes use of the MAC function HMAC based on SHA-256, which is denoted as HMAC-SHA-256$_W$ with 192-bit key W.

The inputs to the Simple Pairing function f3 are

W is 192 bits

$N_1$ is 128 bits

$N_2$ is 128 bits

R is 128 bits

IOcap is 24 bits

$A_1$ is 48 bits

$A_2$ is 48 bits

IOcap is three octets with the most significant octet as the Authentication Requirements parameter, the middle octet as the LMP Out-of-Band Authentication Data parameter, and the least significant octet as the LMP IO capability parameter.

The output of the Simple Pairing *f3* function is:

$f3(W, N_1, N_2, R, IOcap, A_1, A_2) = HMAC\text{-}SHA\text{-}256_W (N_1 \| N_2 \| R \| IOcap \| A_1 \| A_2) / 2^{128}$

The output of *f3* is taken as the 128 most significant (leftmost) bits of the output of HMAC-SHA-256. The check values are computed with function f3. The inputs to f3 are different depending on the different protocols:

| Numeric Comparison | Out-Of-Band | Passkey Entry |
|---|---|---|
| Ea = *f3*(DHKey, Na, Nb, 0, IOcapA, A, B) | Ea = *f3*(DHKey, Na, Nb, rb, IOcapA, A, B) | Ea = *f3*(DHKey, Na20, Nb20, rb, IOcapA, A, B) |
| Eb = *f3*(DHKey, Nb, Na, 0, IOcapB, B, A) | Eb = *f3*(DHKey, Nb, Na, ra, IOcapB, B, A) | Eb = *f3*(DHKey, Nb20, Na20, ra, IOcapB, B, A) |

*Table 7.3: Inputs to f3 for the different protocols*

DHKey denotes the shared secret Diffie-Hellman Key computed as P192(SKa, PKb) by A and as P192(SKb, PKa) by B. Adata denotes the capability data of A and Bdata denotes the capability data of B. In Passkey Entry, the data ra and rb are 6-digit passkey values which are expressed as a 128-bit integer. For instance,

if the 6-digit value of ra is 131313

then R = 0x 00 00 00 00 00 00 00 00 00 00 00 00 00 02 00 f1.

The input A is the BD_ADDR of device A and the input B is the BD_ADDR of device B.

### 7.7.5 The Simple Pairing AMP Key Derivation Function h2

The definition of the Simple Pairing dedicated AMP key derivation function makes use of the MAC function HMAC based on SHA-256, which is denoted as HMAC-SHA-256$_W$ with 256-bit key W.

The inputs to the Simple Pairing function h2 are as follows:

W is 256 bits

keyID is 32 bits

L is 8 bits

The output of the Simple Pairing *h2* function is as follows:

$h2$(W, keyID, L) = HMAC-SHA-256$_W$ (keyID) / $2^{256-(L*8)}$

For each Dedicated AMP, a fixed value of the key identifier KeyID is specified. A master key $MK_{AMP}$ of length $L_{AMP}$, $0 \leq L_{AMP} \leq 32$, is computed as

$MK_{AMP}$ = *h2* (GAMP_LK, $keyID_{AMP}$, $L_{AMP}$).

#### *7.7.5.1 Initial GAMP_LK Creation*

The Generic AMP Link Key (GAMP_LK) shall be created every time the link key (LK) is changed or created using the f2 function (see Section 7.7.3). Whenever the GAMP_LK is created, all dedicated AMP link keys associated with the remote BD_ADDR shall be discarded.

A fixed value of the key identifier KeyID is specified ('gamp'). The string 'gamp' is mapped into keyID using extended ASCII[1] as follows:

**keyID[0]** = 0111 0000 (lsb)
**keyID[1]** = 0110 1101
**keyID[2]** = 0110 0001
**keyID[3]** = 0110 0111
**keyID** = 0x67616d70

The initial GAMP_LK shall be created in the Host using the h2 function.

GAMP_LK = *h2* (W, 'gamp', 32)

Where W = LK || LK

1. (see www.iso.org)ISO_8859-1

### *7.7.5.2 Dedicated AMP Link Key Generation*

Each AMP uses a defined keyID and keyLength. Both values are defined in the Assigned numbers page.

A dedicated AMP link key is computed as

Dedicated_AMP_Link_Key = *h2* (GAMP_LK, keyID, keyLength)

### *7.7.5.3 GAMP_LK Regeneration*

Each time the GAMP_LK is used to generate a valid dedicated AMP link key it shall be regenerated by computing

New_ GAMP_LK = *h2* (GAMP_LK, 'gamp', 32)

and then setting

GAMP_LK = New_GAMP_LK

### *7.7.5.4 Debug AMP Link Key Generation*

When the BR/EDR link key is a debug link key, the dedicated AMP link keys are set equal to the keyLength least significant octets of the Generic AMP link key. When debug link keys are used, the Generic AMP link key shall not be regenerated.

# 8 AMP SECURITY

## 8.1 CREATION OF THE INITIAL GENERIC AMP LINK KEY

AMP security starts any time the BR/EDR link key is created or changed. This may be as the result of the f2 function during Phase 4 of the Secure Simple Pairing process (see Section 7.7.3) or due to a change connection link key procedure. The initial Generic AMP link key, GAMP_LK, is created with the h2 function using the BR/EDR Link Key concatenated together with itself to form a 256-bit value and with the KeyID 'gamp' (see Section 7.7.5.1). The Generic AMP Link Key (GAMP_LK) is stored in the security database alongside the BR/EDR Link Key once pairing has completed.

Whenever the Generic AMP Link Key is created, all Dedicated AMP Link Keys associated with the remote device shall be discarded and any active AMP Physical Links to the remote device shall be disconnected. Note that when a Host initiates a Change Connection Link Key procedure, all active AMP physical links between the two devices will be disconnected.

Since AMP security does not affect the BR/EDR Link Key, backwards compatibility is maintained for devices that support the Generic AMP feature with devices that do not. The Generic AMP Link Key is generated as part of Secure Simple Pairing regardless of whether the devices both support the Generic AMP feature or any AMP in common.

## 8.2 CREATION OF DEDICATED AMP LINK KEYS

When an AMP is used for the first time, a dedicated AMP key is created by the AMP Manager for that AMP type using the new Secure Simple Pairing function h2 and a KeyID for the AMP type. The length of the Dedicated AMP Link Key depends on the AMP Type. If the Dedicated AMP Link Key already exists due to a previous connection the AMP link key is not created and the stored key is reused.

The Dedicated AMP Link Key is sent down to the PAL in the *HCI Create Physical Link* and *HCI Accept Physical Link* commands (see [Vol. 2], Part E Section 7.1.37 and [Vol. 2], Part E Section 7.1.38). Each PAL is responsible for establishing security from the Dedicated AMP Link Key. Note that a Dedicated AMP Link Key may be used for multiple sessions over the same AMP.

Each time a dedicated AMP key is created and the AMP Create Physical Link sequence completes successfully (see [Vol. 3], Part E Section 2.3.3), the Generic AMP Link Key is updated. This is performed using the h2 function with KeyID 'gamp.' If the AMP Create Physical Link sequence does not complete successfully, the dedicated AMP link key shall be discarded and the Generic AMP Link Key shall not be updated.

Figure 8.1The following figure shows an overview of the AMP key generation in the Host.

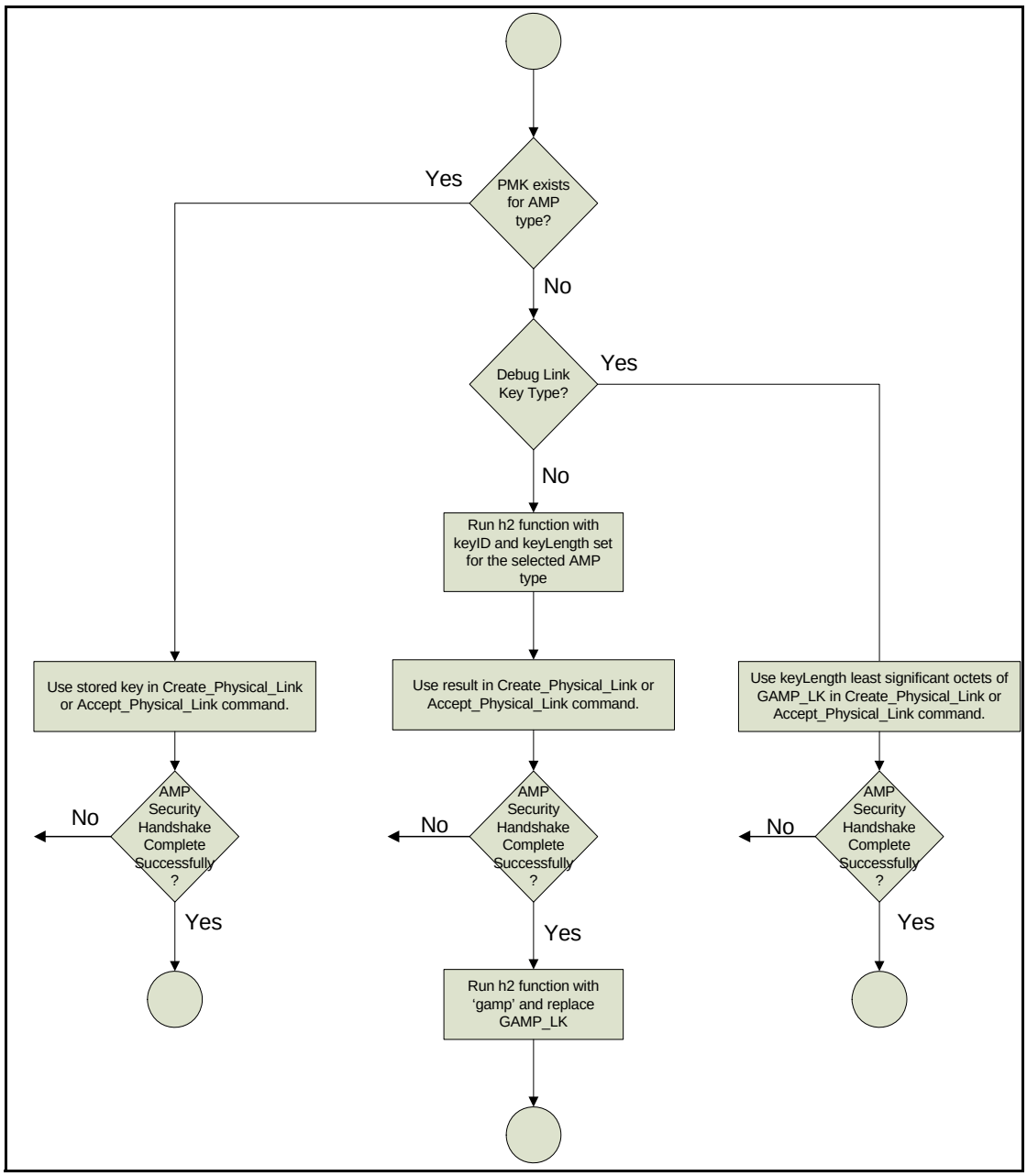



*Figure 8.1: AMP key generation in Host*

Creation of the dedicated AMP keys and updates to the Generic AMP Link Key are the responsibility of the AMP Manager (see [Vol. 3], Part E Section 2.3.3).

## 8.3 DEBUG CONSIDERATIONS

Since the AMP architecture requires encryption to be enabled on each AMP, in order to support sniffers for debug as well as to enable testing over the air, pro-

visions need to be made in the AMP architecture to support debug combination keys. In this document, this is supported in two ways:

- Passing the link key type along with the link key (and length) in the HCI_Create_Physical_Link and HCI_Accept_Physical_Link commands and
- Requiring that the GAMP_LK is used as the dedicated AMP key when the key type is set to "debug combination key."

# 9 LIST OF FIGURES

Figure 3.1: Generation of unit key. When the unit key has been exchanged, the initialization key is discarded in both devices. .....1067
Figure 3.2: Generating a combination key. The old link key (K) is discarded after the exchange of a new combination key has succeeded .....1068
Figure 3.3: Master link key distribution and computation of the corresponding encryption key. .....1071
Figure 4.1: Stream ciphering for Bluetooth with E0. .....1072
Figure 4.2: Functional description of the encryption procedure .....1075
Figure 4.3: Concept of the encryption engine. .....1076
Figure 4.4: Overview of the operation of the encryption engine. Between each start of a packet (TX or RX), the LFSRs are re-initialized. .....1078
Figure 4.5: Arranging the input to the LFSRs. .....1080
Figure 4.6: Distribution of the 128 last generated output symbols within the LFSRs. .....1081
Figure 5.1: Challenge-response for the Bluetooth. .....1082
Figure 5.2: Challenge-response for symmetric key systems. .....1083
Figure 6.1: Flow of data for the computation of E1. .....1086
Figure 6.2: One round in Ar and A'r. .....1088
Figure 6.3: Key scheduling in Ar. .....1089
Figure 6.4: Key generating algorithmE2 and its two modes. Mode 1 is used for unit and combination keys, while mode 2 is used for Kinit and K master. .....1091
Figure 6.5: Generation of the encryption key .....1091
Figure 7.1: Simple Pairing Security Phases .....1092
Figure 7.2: Public Key Exchange Details .....1094
Figure 7.3: Authentication Stage 1 (High Level) .....1094
Figure 7.4: Authentication Stage 1: Numeric Comparison Protocol Details .....1095
Figure 7.5: Authentication Stage 1: Out of Band Details .....1097
Figure 7.6: Authentication Stage 1: Passkey Entry Details .....1099
Figure 7.7: Authentication Stage 2 .....1100
Figure 7.8: Link Key Calculation .....1101
Figure 8.1: AMP key generation in Host .....1109



Specification Volume 3

# SPECIFICATION OF THE *BLUETOOTH* SYSTEM

Experience More



# Core System Package

## [Host volume]

Covered Core Package version:
4.0

Bluetooth®



Ford Ex. 1052

## Revision History

The Revision History is shown in the “Appendix” on page 55 [Vol. 0].

## Contributors

The persons who contributed to this specification are listed in the “Appendix” on page 55 [Vol. 0].

## Web Site

This specification can also be found on the official Bluetooth web site: http://www.bluetooth.com

## Disclaimer and Copyright Notice

The copyright in these specifications is owned by the Promoter Members of Bluetooth SIG, Inc. (“Bluetooth SIG”). Use of these specifications and any related intellectual property (collectively, the “Specification”), is governed by the Promoters Membership Agreement among the Promoter Members and Bluetooth SIG (the “Promoters Agreement”), certain membership agreements between Bluetooth SIG and its Adopter and Associate Members (the “Membership Agreements”) and the Bluetooth Specification Early Adopters Agreements (“1.2 Early Adopters Agreements”) among Early Adopter members of the unincorporated Bluetooth special interest group and the Promoter Members (the “Early Adopters Agreement”). Certain rights and obligations of the Promoter Members under the Early Adopters Agreements have been assigned to Bluetooth SIG by the Promoter Members.

Use of the Specification by anyone who is not a member of Bluetooth SIG or a party to an Early Adopters Agreement (each such person or party, a “Member”), is prohibited. The legal rights and obligations of each Member are governed by their applicable Membership Agreement, Early Adopters Agreement or Promoters Agreement. No license, express or implied, by estoppel or otherwise, to any intellectual property rights are granted herein.

Any use of the Specification not in compliance with the terms of the applicable Membership Agreement, Early Adopters Agreement or Promoters Agreement is prohibited and any such prohibited use may result in termination of the applicable Membership Agreement or Early Adopters Agreement and other liability permitted by the applicable agreement or by applicable law to Bluetooth SIG or any of its members for patent, copyright and/or trademark infringement.

THE SPECIFICATION IS PROVIDED “AS IS” WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, SATISFACTORY QUALITY, OR REASONABLE SKILL OR CARE, OR ANY WAR-

RANTY ARISING OUT OF ANY COURSE OF DEALING, USAGE, TRADE PRACTICE, PROPOSAL, SPECIFICATION OR SAMPLE.

Each Member hereby acknowledges that products equipped with the Bluetooth® technology ("Bluetooth® Products") may be subject to various regulatory controls under the laws and regulations of various governments worldwide. Such laws and regulatory controls may govern, among other things, the combination, operation, use, implementation and distribution of Bluetooth® Products. Examples of such laws and regulatory controls include, but are not limited to, airline regulatory controls, telecommunications regulations, technology transfer controls and health and safety regulations. Each Member is solely responsible for the compliance by their Bluetooth® Products with any such laws and regulations and for obtaining any and all required authorizations, permits, or licenses for their Bluetooth® Products related to such regulations within the applicable jurisdictions. Each Member acknowledges that nothing in the Specification provides any information or assistance in connection with securing such compliance, authorizations or licenses. NOTHING IN THE SPECIFICATION CREATES ANY WARRANTIES, EITHER EXPRESS OR IMPLIED, REGARDING SUCH LAWS OR REGULATIONS.

ALL LIABILITY, INCLUDING LIABILITY FOR INFRINGEMENT OF ANY INTELLECTUAL PROPERTY RIGHTS OR FOR NONCOMPLIANCE WITH LAWS, RELATING TO USE OF THE SPECIFICATION IS EXPRESSLY DISCLAIMED. BY USE OF THE SPECIFICATION, EACH MEMBER EXPRESSLY WAIVES ANY CLAIM AGAINST BLUETOOTH SIG AND ITS PROMOTER MEMBERS RELATED TO USE OF THE SPECIFICATION.

Bluetooth SIG reserves the right to adopt any changes or alterations to the Specification as it deems necessary or appropriate.

Copyright © 1999 - 2010

Ericsson AB,
Lenovo,
Intel Corporation,
Microsoft Corporation,
Motorola, Inc.,
Nokia Corporation,
Toshiba Corporation

*Third-party brands and names are the property of their respective owners.

# CONTENTS

**Part A**
**LOGICAL LINK CONTROL AND ADAPTATION PROTOCOL SPECIFICATION**

**1 Introduction ..... 30**
1.1 L2CAP Features ..... 30
1.2 Assumptions ..... 33
1.3 Scope ..... 34
1.4 Terminology ..... 34

**2 General Operation ..... 38**
2.1 Channel Identifiers ..... 38
2.2 Operation Between Devices ..... 38
2.3 Operation Between Layers ..... 40
2.4 Modes of Operation ..... 41
2.5 Mapping Channels to Logical Links ..... 43

**3 Data Packet Format ..... 44**
3.1 Connection-oriented Channels in Basic L2CAP Mode ..... 44
3.2 Connectionless Data Channel in Basic L2CAP Mode ..... 45
3.3 Connection-oriented Channel in Retransmission/Flow Control/ Streaming Modes ..... 45
3.3.1 L2CAP header fields ..... 46
3.3.2 Control field (2 or 4 octets) ..... 48
3.3.3 L2CAP SDU Length Field (2 octets) ..... 50
3.3.4 Information Payload Field ..... 50
3.3.5 Frame Check Sequence (2 octets) ..... 51
3.3.6 Invalid Frame Detection ..... 52
3.3.7 Invalid Frame Detection Algorithm ..... 52

**4 Signaling Packet Formats ..... 54**
4.1 Command Reject (code 0x01) ..... 56
4.2 Connection Request (code 0x02) ..... 58
4.3 Connection Response (code 0x03) ..... 59
4.4 Configuration Request (code 0x04) ..... 60
4.5 Configuration Response (code 0x05) ..... 62
4.6 Disconnection Request (code 0x06) ..... 64
4.7 Disconnection Response (code 0x07) ..... 65
4.8 Echo Request (code 0x08) ..... 66
4.9 Echo Response (code 0x09) ..... 66
4.10 Information Request (code 0x0A) ..... 66
4.11 Information Response (code 0x0B) ..... 67

4.12 Extended Feature Mask ..... 69
4.13 Fixed Channels Supported ..... 70
4.14 Create Channel Request (code 0x0C) ..... 70
4.15 Create Channel Response (code 0x0D) ..... 71
4.16 Move Channel Request (code 0x0E) ..... 73
4.17 Move Channel Response (code 0x0F) ..... 74
4.18 Move Channel Confirmation (code 0x10) ..... 75
4.19 Move Channel Confirmation Response (code 0x11) ..... 76
4.20 Connection Parameter Update Request (code 0X12) ..... 76
4.21 Connection Parameter Update Response (code 0X13) ..... 78

**5 Configuration Parameter Options ..... 79**
5.1 Maximum Transmission Unit (MTU) ..... 79
5.2 Flush Timeout Option ..... 81
5.3 Quality of Service (QoS) Option ..... 82
5.4 Retransmission and Flow Control Option ..... 86
5.5 Frame Check Sequence (FCS) Option ..... 91
5.6 Extended Flow Specification Option ..... 92
5.7 Extended Window Size Option ..... 96

**6 State Machine ..... 98**
6.1 General rules for the state machine: ..... 98
6.1.1 CLOSED state ..... 99
6.1.2 WAIT_CONNECT_RSP state ..... 101
6.1.3 WAIT_CONNECT state ..... 102
6.1.4 CONFIG state ..... 102
6.1.5 OPEN state ..... 108
6.1.6 WAIT_DISCONNECT state ..... 109
6.1.7 WAIT_CREATE_RSP state ..... 110
6.1.8 WAIT_CREATE state ..... 110
6.1.9 WAIT_MOVE_RSP state ..... 111
6.1.10 WAIT_MOVE state ..... 113
6.1.11 WAIT_MOVE_CONFIRM state ..... 114
6.1.12 WAIT_CONFIRM_RSP state ..... 114
6.2 Timers events ..... 114
6.2.1 RTX ..... 114
6.2.2 ERTX ..... 116

**7 General Procedures ..... 120**
7.1 Configuration Process ..... 120
7.1.1 Request Path ..... 121
7.1.2 Response pPath ..... 121
7.1.3 Lockstep Configuration Process ..... 122

7.1.4 Standard Configuration Process....125
7.2 Fragmentation and Recombination....126
7.2.1 Fragmentation of L2CAP PDUs....126
7.2.2 Recombination of L2CAP PDUs....128
7.3 Encapsulation of SDUs....128
7.3.1 Segmentation of L2CAP SDUs....129
7.3.2 Reassembly of L2CAP SDUs....129
7.3.3 Segmentation and fragmentation....130
7.4 Delivery of Erroneous L2CAP SDUs....132
7.5 Operation with Flushing On ACL-U Logical Links....132
7.6 Connectionless Data Channel....133
7.7 Operation Collision Resolution....135
7.8 Aggregating Best Effort Extended Flow Specifications....135
7.9 Prioritizing Data over HCI....136
7.10 Supporting Extended Flow Specification for BR/EDR And BR/EDR/LE Controllers....137

**8 Procedures for Flow Control and Retransmission....139**

8.1 Information Retrieval....139
8.2 Function of PDU Types for Flow Control and Retransmission....139
8.2.1 Information frame (I-frame)....139
8.2.2 Supervisory Frame (S-frame)....139
8.3 Variables and Sequence Numbers....140
8.3.1 Sending peer....141
8.3.2 Receiving peer....142
8.4 Retransmission Mode....144
8.4.1 Transmitting frames....144
8.4.2 Receiving I-frames....146
8.4.3 I-frames pulled by the SDU reassembly function....146
8.4.4 Sending and receiving acknowledgements....147
8.4.5 Receiving REJ frames....148
8.4.6 Waiting acknowledgements....149
8.4.7 Exception conditions....149
8.5 Flow Control Mode....150
8.5.1 Transmitting I-frames....151
8.5.2 Receiving I-frames....151
8.5.3 I-frames pulled by the SDU reassembly function....152
8.5.4 Sending and receiving acknowledgements....152
8.5.5 Waiting acknowledgements....152
8.5.6 Exception conditions....153
8.6 Enhanced Retransmission Mode....154

8.6.1 Function Of PDU Types ..... 155
8.6.2 Rules For Timers ..... 156
8.6.3 General Rules for the State Machine ..... 159
8.6.4 State Diagram ..... 160
8.6.5 States Tables ..... 161
8.7 Streaming Mode ..... 184
8.7.1 Transmitting I-frames ..... 185
8.7.2 Receiving I-frames ..... 185
8.7.3 Exception Conditions ..... 185
**9 Procedure for AMP Channel Creation and Handling ..... 186**
9.1 Create Channel ..... 186
9.2 Move Channel ..... 189
9.2.1 Move Channel Protocol Procedure with Enhanced Retransmission Mode ..... 190
9.2.2 Move Channel Protocol Procedure with Streaming Mode (Initiator is Data Source) ..... 193
9.2.3 Move Channel Protocol Procedure with Streaming Mode (Initiator is Data Sink) ..... 194
9.3 Disconnect Channel ..... 197
**10 List of Figures ..... 198**
**11 List of Tables ..... 200**

**Part B**
**SERVICE DISCOVERY PROTOCOL (SDP) SPECIFICATION**

**1 Introduction ..... 209**
1.1 General Description ..... 209
1.2 Motivation ..... 209
1.3 Requirements ..... 209
1.4 Non-requirements and Deferred Requirements ..... 210
1.5 Conventions ..... 210
1.5.1 Bit And Byte Ordering Conventions ..... 210
**2 Overview ..... 211**
2.1 SDP Client-Server Interaction ..... 211
2.2 Service Record ..... 212
2.3 Service Attribute ..... 213
2.3.1 Attribute ID ..... 214
2.3.2 Attribute Value ..... 215
2.4 Service Class ..... 215
2.4.1 A Printer Service Class Example ..... 215
2.5 Searching for Services ..... 216

2.5.1 UUID....216
2.5.2 Service Search Patterns....217
2.6 Browsing for Services....217
2.6.1 Example Service Browsing Hierarchy....218
**3 Data Representation....220**
3.1 Data Element....220
3.2 Data Element Type Descriptor....220
3.3 Data Element Size Descriptor....221
3.4 Data Element Examples....222
**4 Protocol Description....223**
4.1 Transfer Byte Order....223
4.2 Protocol Data Unit Format....223
4.3 Partial Responses and Continuation State....225
4.4 Error Handling....225
4.4.1 SDP_ErrorResponse PDU....226
4.5 ServiceSearch Transaction....227
4.5.1 SDP_ServiceSearchRequest PDU....227
4.5.2 SDP_ServiceSearchResponse PDU....228
4.6 ServiceAttribute Transaction....230
4.6.1 SDP_ServiceAttributeRequest PDU....230
4.6.2 SDP_ServiceAttributeResponse PDU....232
4.7 ServiceSearchAttribute Transaction....233
4.7.1 SDP_ServiceSearchAttributeRequest PDU....233
4.7.2 SDP_ServiceSearchAttributeResponse PDU....235
**5 Service Attribute Definitions....237**
5.1 Universal Attribute Definitions....237
5.1.1 ServiceRecordHandle Attribute....237
5.1.2 ServiceClassIDList Attribute....237
5.1.3 ServiceRecordState Attribute....238
5.1.4 ServiceID Attribute....238
5.1.5 ProtocolDescriptorList Attribute....239
5.1.6 AdditionalProtocolDescriptorList Attribute....240
5.1.7 BrowseGroupList Attribute....241
5.1.8 LanguageBaseAttributeIDList Attribute....241
5.1.9 ServiceInfoTimeToLive Attribute....242
5.1.10 ServiceAvailability Attribute....243
5.1.11 BluetoothProfileDescriptorList Attribute....243
5.1.12 DocumentationURL Attribute....244
5.1.13 ClientExecutableURL Attribute....244

5.1.14 IconURL Attribute ..... 245
5.1.15 ServiceName Attribute ..... 245
5.1.16 ServiceDescription Attribute ..... 246
5.1.17 ProviderName Attribute ..... 246
5.1.18 Reserved Universal Attribute IDs ..... 246
5.2 ServiceDiscoveryServer Service Class Attribute Definitions ..... 246
5.2.1 ServiceRecordHandle Attribute ..... 246
5.2.2 ServiceClassIDList Attribute ..... 247
5.2.3 VersionNumberList Attribute ..... 247
5.2.4 ServiceDatabaseState Attribute ..... 247
5.2.5 Reserved Attribute IDs ..... 248
5.3 BrowseGroupDescriptor Service Class Attribute Definitions ..... 248
5.3.1 ServiceClassIDList Attribute ..... 248
5.3.2 GroupID Attribute ..... 249
5.3.3 Reserved Attribute IDs ..... 249
**6 Security ..... 250**

**Part C**
**GENERIC ACCESS PROFILE**

**1 Introduction ..... 274**
1.1 Scope ..... 274
1.2 Symbols and Conventions ..... 275
1.2.1 Requirement Status Symbols ..... 275
1.2.2 Signaling diagram conventions ..... 276
1.2.3 Notation for Timers and Counters ..... 276
**2 Profile Overview ..... 277**
2.1 Profile Stack ..... 277
2.2 Profile Roles ..... 277
2.2.1 Roles when Operating over BR/EDR Physical Channel ..... 277
2.2.2 Roles when Operating over an LE Physical Channel ..... 278
2.3 User Requirements and Scenarios ..... 280
2.4 Profile Fundamentals ..... 281
2.5 Conformance ..... 281
2.6 Other Requirements ..... 281
**3 User Interface Aspects ..... 282**
3.1 The User Interface Level ..... 282
3.2 Representation of Bluetooth Parameters ..... 282
3.2.1 Bluetooth Device Address (BD_ADDR) ..... 282

3.2.2 Bluetooth Device Name (the user-friendly name) .....283
3.2.3 Bluetooth Passkey (Bluetooth PIN) .....284
3.2.4 Class of Device .....286
3.3 Pairing .....286
**4 Modes .....287**
4.1 Discoverability Modes .....287
4.1.1 Non-discoverable Mode .....288
4.1.2 Limited Discoverable Mode .....288
4.1.3 General Discoverable Mode .....289
4.2 Connectability Modes .....291
4.2.1 Non-connectable Mode .....291
4.2.2 Connectable Mode .....291
4.3 Bondable Modes .....293
4.3.1 Non-bondable Mode .....293
4.3.2 Bondable Mode .....293
**5 Security Aspects .....295**
5.1 Authentication .....295
5.1.1 Purpose .....295
5.1.2 Term on UI level .....295
5.1.3 Procedure .....296
5.1.4 Conditions .....296
5.2 Security Modes .....296
5.2.1 Legacy Security Modes .....297
5.2.2 Security Mode 4 (service level enforced security) .....299
**6 Idle Mode Procedures .....314**
6.1 General Inquiry .....314
6.1.1 Purpose .....314
6.1.2 Term on UI level .....314
6.1.3 Description .....315
6.1.4 Conditions .....315
6.2 Limited Inquiry .....315
6.2.1 Purpose .....315
6.2.2 Term on UI level .....316
6.2.3 Description .....316
6.2.4 Conditions .....316
6.3 Name Discovery .....317
6.3.1 Purpose .....317
6.3.2 Term on UI level .....317

6.3.3 Description ..... 317
6.3.4 Conditions ..... 318
6.4 Device Discovery ..... 318
6.4.1 Purpose ..... 318
6.4.2 Term on UI Level ..... 318
6.4.3 Description ..... 319
6.4.4 Conditions ..... 319
6.5 Bonding ..... 319
6.5.1 Purpose ..... 319
6.5.2 Term on UI level ..... 319
6.5.3 Description ..... 320
6.5.4 Conditions ..... 321
**7 Establishment Procedures ..... 323**
7.1 Link Establishment ..... 323
7.1.1 Purpose ..... 323
7.1.2 Term on UI Level ..... 323
7.1.3 Description ..... 323
7.1.4 Conditions ..... 325
7.2 Channel Establishment ..... 326
7.2.1 Purpose ..... 326
7.2.2 Term on UI level ..... 326
7.2.3 Description ..... 326
7.2.4 Conditions ..... 327
7.3 Connection Establishment ..... 328
7.3.1 Purpose ..... 328
7.3.2 Term on UI level ..... 328
7.3.3 Description ..... 328
7.3.4 Conditions ..... 329
7.4 Establishment of Additional Connection ..... 329
**8 Extended Inquiry Response Data Format ..... 330**
8.1 EIR Data Type Definitions ..... 331
8.1.1 Service Class UUIDs ..... 331
8.1.2 Local Name ..... 332
8.1.3 Flags ..... 332
8.1.4 Manufacturer Specific Data ..... 332
8.1.5 TX Power Level ..... 333
8.1.6 Secure Simple Pairing Out of Band (OOB) ..... 333
8.2 Example Extended Inquiry Response ..... 333

**9 Operational Modes and Procedures For Use On LE Physical Channels ..... 335**
9.1 Broadcast Mode and Observation Procedure ..... 335
9.1.1 Broadcast Mode ..... 335
9.1.2 Observation Procedure ..... 336
9.2 Discovery Modes and Procedures ..... 337
9.2.1 Requirements ..... 338
9.2.2 Non-Discoverable Mode ..... 338
9.2.3 Limited Discoverable Mode ..... 338
9.2.4 General Discoverable Mode ..... 340
9.2.5 Limited Discovery Procedure ..... 341
9.2.6 General Discovery Procedure ..... 343
9.2.7 Name Discovery Procedure ..... 344
9.3 Connection Modes and Procedures ..... 344
9.3.1 Requirements ..... 346
9.3.2 Non-Connectable Mode ..... 346
9.3.3 Directed Connectable Mode ..... 347
9.3.4 Undirected Connectable Mode ..... 349
9.3.5 Auto Connection Establishment Procedure ..... 351
9.3.6 General Connection Establishment Procedure ..... 354
9.3.7 Selective Connection Establishment Procedure ..... 356
9.3.8 Direct Connection Establishment Procedure ..... 357
9.3.9 Connection Parameter Update Procedure ..... 358
9.3.10 Terminate Connection Procedure ..... 358
9.4 Bonding Modes and Procedures ..... 359
9.4.1 Requirements ..... 359
9.4.2 Non-Bondable Mode ..... 359
9.4.3 Bondable Mode ..... 360
9.4.4 Bonding Procedure ..... 360
**10 LE Security Aspects ..... 362**
10.1 Requirements ..... 362
10.2 LE Security Modes ..... 362
10.2.1 LE Security Mode 1 ..... 362
10.2.2 LE Security Mode 2 ..... 363
10.2.3 Mixed Security Modes Requirements ..... 363
10.3 Authentication Procedure ..... 363
10.3.1 Receiving a Service Request ..... 364
10.3.2 Initiating a Service Request ..... 365
10.4 Data Signing ..... 368

10.4.1 Connection Data Signing Procedure ..... 368
10.4.2 Authenticate Signed Data Procedure ..... 368
10.5 Authorization Procedure ..... 369
10.6 Encryption Procedure ..... 369
10.7 Privacy Feature ..... 369
10.7.1 Privacy Feature in a Peripheral ..... 370
10.7.2 Privacy Feature in a Central ..... 371
10.8 Random Device Address ..... 371
10.8.1 Static Address ..... 371
10.8.2 Private address ..... 372
**11 Advertising and Scan Response Data Format ..... 375**
11.1 AD Type Definitions ..... 375
11.1.1 Service UUIDs ..... 376
11.1.2 Local Name ..... 376
11.1.3 Flags ..... 376
11.1.4 Manufacturer Specific Data ..... 376
11.1.5 TX Power Level ..... 377
11.1.6 Security Manager Out of Band (OOB) ..... 377
11.1.7 Security Manager TK Value ..... 377
11.1.8 Slave Connection Interval Range ..... 377
11.1.9 Service Solicitation ..... 377
11.1.10 Service Data ..... 377
11.2 Example Advertising Data ..... 378
**12 GAP Characteristics for Low Energy ..... 379**
12.1 Device Name Characteristic ..... 379
12.2 Appearance Characteristic ..... 380
12.3 Peripheral Privacy Flag Characteristic ..... 380
12.4 Reconnection Address Characteristic ..... 380
12.5 Peripheral Preferred Connection Parameters Characteristic ..... 381
**13 BR/EDR/LE Operation Modes and Procedure ..... 383**
13.1 Modes ..... 383
13.1.1 Discoverability ..... 384
13.1.2 Connectability ..... 385
13.1.3 Bondable Modes ..... 385
13.2 Idle Mode Procedures ..... 385
13.2.1 General Discovery Procedure ..... 386
13.2.2 Limited Discovery Procedure ..... 386
13.2.3 Device Discovery Procedure ..... 387
13.2.4 Name Discovery ..... 387

13.3 Establishment Procedures ....388
13.3.1 Link Establishment ....388
13.4 SDP Interoperability Requirements ....388
**14 BR/EDR/LE Security Aspects ....390**
**15 Definitions ....391**
15.1 General Definitions ....391
15.2 Connection-related Definitions ....391
15.3 Device-related Definitions ....392
15.4 Procedure-related Definitions ....393
15.5 Security-related Definitions ....393
**16 Appendix A (Normative): Timers and Constants ....395**
**17 Appendix B (Informative): Information Flows of Related Procedures ....398**
17.1 LMP – Authentication ....398
17.2 LMP – Pairing ....399
17.3 Service Discovery ....400
17.4 Generating a Resolvable Private Address ....400
17.5 Resolving a Resolvable Private Address ....400
**18 Appendix C (Normative): EIR and AD Formats ....401**
18.1 Flags ....401
18.2 Service ....402
18.3 Local Name ....402
18.4 TX Power Level ....402
18.5 Simple Pairing Optional OOB Tags ....403
18.6 Security Manager TK Value ....403
18.7 Security Manager OOB Flags ....403
18.8 Slave Connection Interval Range ....404
18.9 Service Solicitation ....404
18.10 Service Data ....404
18.11 Manufacturer Specific Data ....405
**19 References ....406**

**Part D**
**TEST SUPPORT**

**1 Test Methodology ....411**
1.1 BR/EDR Test Scenarios ....411
1.1.1 Test setup ....411
1.1.2 Transmitter Test ....412
1.1.3 LoopBack test ....416
1.1.4 Pause test ....420

1.2 AMP Test Scenarios ..... 421
1.2.1 Methodology Overview ..... 421
1.2.2 Control and Configuration ..... 422
1.2.3 AMP Test Manager ..... 423
1.2.4 Test Commands/Events Format ..... 424
1.2.5 AMP Test Manager Commands/Events ..... 425
1.3 References ..... 429
**2 Test Control Interface (TCI) ..... 430**
2.1 Introduction ..... 430
2.1.1 Terms Used ..... 430
2.1.2 Usage of the Interface ..... 430
2.2 TCI Configurations ..... 431
2.2.1 Bluetooth RF Requirements ..... 431
2.2.2 Bluetooth protocol requirements ..... 432
2.2.3 Bluetooth Profile Requirements ..... 433
2.3 TCI Configuration and Usage ..... 434
2.3.1 Transport Layers ..... 434
2.3.2 Baseband and Link Manager Qualification ..... 435
2.3.3 HCI Qualification ..... 437

**Part E**
**AMP MANAGER PROTOCOL SPECIFICATION**

**1 Introduction ..... 442**
1.1 General Description ..... 442
**2 General Operation ..... 443**
2.1 Basic Capabilities ..... 443
2.2 AMP Manager Channel Over L2CAP ..... 444
2.3 Using the AMP Manager Protocol ..... 445
2.3.1 Discovering a Remote AMP Manager ..... 445
2.3.2 Discovering Available Controllers on a Remote Device ..... 445
2.3.3 Creation of AMP Physical Links ..... 446
2.4 Controller IDs ..... 447
2.5 Controller Types ..... 447
**3 Protocol Description ..... 448**
3.1 Packet Formats ..... 448
3.2 AMP Command Reject (Code 0x01) ..... 449
3.3 AMP Discover Request (Code 0x02) ..... 450
3.4 AMP Discover Response (Code 0x03) ..... 451

3.5 AMP Change Notify (Code 0x04) ..... 455
3.6 AMP Change Response (Code 0x05) ..... 456
3.7 AMP Get Info Request (Code 0x06) ..... 456
3.8 AMP Get Info Response (Code 0x07) ..... 456
3.9 AMP Get AMP Assoc Request (Code 0x08) ..... 458
3.10 AMP Get AMP Assoc Response (Code 0x09) ..... 459
3.11 AMP Create Physical Link Request (Code 0x0A) ..... 460
3.12 AMP Create Physical Link Response (Code 0x0B) ..... 461
3.13 AMP Disconnect Physical Link Request (Code 0x0C) ..... 462
3.14 AMP Disconnect Physical Link Response (Code 0x0D) ..... 463
3.15 Create Physical Link Collision Resolution ..... 464
3.16 Response Timeout ..... 465
3.17 Unexpected BR/EDR Physical Link Disconnect ..... 465

**Part F**
**ATTRIBUTE PROTOCOL (ATT)**

**1 Introduction ..... 473**
1.1 Scope ..... 473
1.2 Conformance ..... 473

**2 Protocol Overview ..... 474**

**3 Protocol Requirements ..... 475**
3.1 Introduction ..... 475
3.2 Basic Concepts ..... 475
3.2.1 Attribute Type ..... 475
3.2.2 Attribute Handle ..... 476
3.2.3 Attribute Handle Grouping ..... 476
3.2.4 Attribute Value ..... 476
3.2.5 Attribute Permissions ..... 477
3.2.6 Control-Point Attributes ..... 478
3.2.7 Protocol Methods ..... 478
3.2.8 Exchanging MTU Size ..... 479
3.2.9 Long Attribute Values ..... 479
3.2.10 Atomic Operations ..... 479
3.3 Attribute PDU ..... 480
3.3.1 Attribute PDU Format ..... 481
3.3.2 Sequential Protocol ..... 482
3.3.3 Transaction ..... 482
3.4 Attribute Protocol PDUs ..... 483
3.4.1 Error Handling ..... 483
3.4.2 MTU Exchange ..... 485

3.4.3 Find Information ..... 487
3.4.4 Reading Attributes ..... 491
3.4.5 Writing Attributes ..... 501
3.4.6 Queued Writes ..... 505
3.4.7 Server Initiated ..... 509
3.4.8 Attribute Opcode Summary ..... 510
3.4.9 Attribute PDU Response Summary ..... 513

**4 Security Considerations ..... 516**

**5 Acronyms and Abbreviations ..... 517**

**Part G**
**GENERIC ATTRIBUTE PROFILE (GATT)**

**1 Introduction ..... 524**
1.1 Scope ..... 524
1.2 Profile Dependency ..... 524
1.3 Conformance ..... 524
1.4 Bluetooth Specification Release Compatibility ..... 525
1.5 Conventions ..... 525

**2 Profile Overview ..... 526**
2.1 Protocol Stack ..... 526
2.2 Configurations and Roles ..... 526
2.3 User Requirements and Scenarios ..... 527
2.4 Profile Fundamentals ..... 528
2.5 Attribute Protocol ..... 528
2.5.1 Overview ..... 528
2.5.2 Attribute Caching ..... 529
2.5.3 Attribute Grouping ..... 531
2.5.4 UUIDs ..... 531
2.6 GATT Profile Hierarchy ..... 531
2.6.1 Overview ..... 531
2.6.2 Service ..... 532
2.6.3 Included Services ..... 533
2.6.4 Characteristic ..... 533
2.7 Configured Broadcast ..... 533

**3 Service Interoperability Requirements ..... 535**
3.1 Service Definition ..... 535
3.2 Include Definition ..... 536
3.3 Characteristic Definition ..... 536
3.3.1 Characteristic Declaration ..... 537

3.3.2 Characteristic Value Declaration ....539
3.3.3 Characteristic Descriptor Declarations ....539
3.4 Summary of GATT Profile Attribute Types ....547
**4 GATT Feature Requirements ....548**
4.1 Overview ....548
4.2 Feature Support and Procedure Mapping ....548
4.3 Server Configuration ....550
4.3.1 Exchange MTU ....550
4.4 Primary Service Discovery ....551
4.4.1 Discover All Primary Services ....551
4.4.2 Discover Primary Service by Service UUID ....552
4.5 Relationship Discovery ....554
4.5.1 Find Included Services ....554
4.6 Characteristic Discovery ....556
4.6.1 Discover All Characteristics of a Service ....556
4.6.2 Discover Characteristics by UUID ....557
4.7 Characteristic Descriptor Discovery ....558
4.7.1 Discover All Characteristic Descriptors ....558
4.8 Characteristic Value Read ....560
4.8.1 Read Characteristic Value ....560
4.8.2 Read Using Characteristic UUID ....560
4.8.3 Read Long Characteristic Values ....561
4.8.4 Read Multiple Characteristic Values ....562
4.9 Characteristic Value Write ....563
4.9.1 Write Without Response ....563
4.9.2 Signed Write Without Response ....564
4.9.3 Write Characteristic Value ....565
4.9.4 Write Long Characteristic Values ....566
4.9.5 Reliable Writes ....567
4.10 Characteristic Value Notification ....569
4.10.1 Notifications ....570
4.11 Characteristic Value Indications ....570
4.11.1 Indications ....570
4.12 Characteristic Descriptors ....571
4.12.1 Read Characteristic Descriptors ....571
4.12.2 Read Long Characteristic Descriptors ....572
4.12.3 Write Characteristic Descriptors ....573
4.12.4 Write Long Characteristic Descriptors ....574
4.13 GATT Procedure Mapping to ATT Protocol Opcodes ....575

4.14 Procedure Timeouts ..... 578

**5 L2CAP Interoperability Requirements ..... 579**

5.1 BR/EDR L2CAP Interoperability Requirements ..... 579

5.1.1 ATT_MTU ..... 579

5.1.2 BR/EDR Channel Requirements ..... 579

5.1.3 BR/EDR Channel Establishment Collisions ..... 579

5.2 LE L2CAP Interoperability Requirements ..... 580

5.2.1 ATT_MTU ..... 580

5.2.2 LE Channel Requirements ..... 580

**6 GAP Interoperability Requirements ..... 582**

6.1 BR/EDR GAP Interoperability Requirements ..... 582

6.1.1 Connection Establishment ..... 582

6.2 LE GAP Interoperability Requirements ..... 582

6.2.1 Connection Establishment ..... 582

6.2.2 Profile Roles ..... 582

6.3 Disconnected Events ..... 582

6.3.1 Notifications and Indications While Disconnected ..... 582

**7 Defined Generic Attribute Profile Service ..... 584**

7.1 Service Changed ..... 584

**8 Security Considerations ..... 586**

8.1 Authentication Requirements ..... 586

8.2 Authorization Requirements ..... 587

**9 SDP Interoperability Requirements ..... 588**

**10 References ..... 589**

**11 Appendix: Example Attribute Server Attributes ..... 590**

**Part H**
**SECURITY MANAGER SPECIFICATION**

**1 Introduction ..... 598**

1.1 Scope ..... 598

1.2 Conventions ..... 598

1.2.1 Bit and Byte Ordering Conventions ..... 598

**2 Security Manager ..... 599**

2.1 Introduction ..... 599

2.2 Cryptographic Toolbox ..... 600

2.2.1 Security function *e* ..... 600

2.2.2 Random Address Hash function *ah* ..... 600

2.2.3 Confirm value generation function c1 ..... 601

2.2.4 Key generation function *s1* ..... 602
2.3 Pairing Methods ..... 603
2.3.1 Security Properties ..... 604
2.3.2 IO Capabilities ..... 605
2.3.3 OOB Authentication Data ..... 606
2.3.4 Encryption Key Size ..... 606
2.3.5 Pairing Algorithms ..... 607
2.3.6 Repeated Attempts ..... 611
2.4 Security in Bluetooth low energy ..... 612
2.4.1 Definition of Keys and Values ..... 612
2.4.2 Generation of Keys ..... 612
2.4.3 Distribution of Keys ..... 614
2.4.4 Encrypted Session Setup ..... 615
2.4.5 Signing Algorithm ..... 616
2.4.6 Slave Security Request ..... 617
**3 Security Manager Protocol ..... 620**
3.1 Introduction ..... 620
3.2 Security Manager Channel over L2CAP ..... 620
3.3 Command Format ..... 620
3.4 SMP Timeout ..... 621
3.5 Pairing Methods ..... 622
3.5.1 Pairing Request ..... 622
3.5.2 Pairing Response ..... 624
3.5.3 Pairing Confirm ..... 625
3.5.4 Pairing Random ..... 626
3.5.5 Pairing Failed ..... 627
3.6 Security in Bluetooth low energy ..... 628
3.6.1 Key Distribution ..... 628
3.6.2 Encryption Information ..... 630
3.6.3 Master Identification ..... 631
3.6.4 Identity Information ..... 631
3.6.5 Identity Address Information ..... 632
3.6.6 Signing Information ..... 633
3.6.7 Security Request ..... 633
**4 References ..... 635**
**5 Appendices ..... 638**
5.1 Appendix A – EDIV and Rand Generation ..... 638
5.1.1 EDIV Masking ..... 638
5.2 Appendix B – Key Management ..... 639

5.2.1 Database Lookup ..... 640
5.2.2 Key Hierarchy ..... 640
5.3 Message Sequence Charts ..... 643
5.3.1 Phase 1: Pairing Feature Exchange ..... 643
5.3.2 Phase 2: Authenticating and Encrypting ..... 644
5.3.3 Phase 3: Transport Specific Key Distribution ..... 647
5.3.4 Security Re-established using Previously Distributed LTK ..... 648
5.3.5 Failure Conditions ..... 649

# LOGICAL LINK CONTROL AND ADAPTATION PROTOCOL SPECIFICATION

***The Bluetooth logical link control and adaptation protocol (L2CAP) supports higher level protocol multiplexing, packet segmentation and reassembly, and the conveying of quality of service information. The protocol state machine, packet format, and composition are described in this document.***


Ford Ex. 1052

# CONTENTS

**1 Introduction ..... 30**
1.1 L2CAP Features ..... 30
1.2 Assumptions ..... 33
1.3 Scope ..... 34
1.4 Terminology ..... 34
**2 General Operation ..... 38**
2.1 Channel Identifiers ..... 38
2.2 Operation Between Devices ..... 38
2.3 Operation Between Layers ..... 40
2.4 Modes of Operation ..... 41
2.5 Mapping Channels to Logical Links ..... 43
**3 Data Packet Format ..... 44**
3.1 Connection-oriented Channels in Basic L2CAP Mode ..... 44
3.2 Connectionless Data Channel in Basic L2CAP Mode ..... 45
3.3 Connection-oriented Channel in Retransmission/Flow Control/ Streaming Modes45
3.3.1 L2CAP header fields ..... 46
3.3.2 Control field (2 or 4 octets) ..... 48
3.3.3 L2CAP SDU Length Field (2 octets) ..... 50
3.3.4 Information Payload Field ..... 50
3.3.5 Frame Check Sequence (2 octets) ..... 51
3.3.6 Invalid Frame Detection ..... 52
3.3.7 Invalid Frame Detection Algorithm ..... 52
**4 Signaling Packet Formats ..... 54**
4.1 Command Reject (code 0x01) ..... 56
4.2 Connection Request (code 0x02) ..... 58
4.3 Connection Response (code 0x03) ..... 59
4.4 Configuration Request (code 0x04) ..... 60
4.5 Configuration Response (code 0x05) ..... 62
4.6 Disconnection Request (code 0x06) ..... 64
4.7 Disconnection Response (code 0x07) ..... 65
4.8 Echo Request (code 0x08) ..... 66
4.9 Echo Response (code 0x09) ..... 66
4.10 Information Request (code 0x0A) ..... 66
4.11 Information Response (code 0x0B) ..... 67
4.12 Extended Feature Mask ..... 69
4.13 Fixed Channels Supported ..... 70
4.14 Create Channel Request (code 0x0C) ..... 70

4.15 Create Channel Response (code 0x0D) ..... 71
4.16 Move Channel Request (code 0x0E) ..... 73
4.17 Move Channel Response (code 0x0F) ..... 74
4.18 Move Channel Confirmation (code 0x10) ..... 75
4.19 Move Channel Confirmation Response (code 0x11) ..... 76
4.20 Connection Parameter Update Request (code 0X12) ..... 76
4.21 Connection Parameter Update Response (code 0X13) ..... 78

**5 Configuration Parameter Options ..... 79**
5.1 Maximum Transmission Unit (MTU) ..... 79
5.2 Flush Timeout Option ..... 81
5.3 Quality of Service (QoS) Option ..... 82
5.4 Retransmission and Flow Control Option ..... 86
5.5 Frame Check Sequence (FCS) Option ..... 91
5.6 Extended Flow Specification Option ..... 92
5.7 Extended Window Size Option ..... 96

**6 State Machine ..... 98**
6.1 General rules for the state machine: ..... 98
6.1.1 CLOSED state ..... 99
6.1.2 WAIT_CONNECT_RSP state ..... 101
6.1.3 WAIT_CONNECT state ..... 102
6.1.4 CONFIG state ..... 102
6.1.5 OPEN state ..... 108
6.1.6 WAIT_DISCONNECT state ..... 109
6.1.7 WAIT_CREATE_RSP state ..... 110
6.1.8 WAIT_CREATE state ..... 110
6.1.9 WAIT_MOVE_RSP state ..... 111
6.1.10 WAIT_MOVE state ..... 113
6.1.11 WAIT_MOVE_CONFIRM state ..... 114
6.1.12 WAIT_CONFIRM_RSP state ..... 114
6.2 Timers events ..... 114
6.2.1 RTX ..... 114
6.2.2 ERTX ..... 116

**7 General Procedures ..... 120**
7.1 Configuration Process ..... 120
7.1.1 Request Path ..... 121
7.1.2 Response pPath ..... 121
7.1.3 Lockstep Configuration Process ..... 122
7.1.4 Standard Configuration Process ..... 125
7.2 Fragmentation and Recombination ..... 126

7.2.1 Fragmentation of L2CAP PDUs ....126
7.2.2 Recombination of L2CAP PDUs....128
7.3 Encapsulation of SDUs ....128
7.3.1 Segmentation of L2CAP SDUs ....129
7.3.2 Reassembly of L2CAP SDUs....129
7.3.3 Segmentation and fragmentation ....130
7.4 Delivery of Erroneous L2CAP SDUs....132
7.5 Operation with Flushing On ACL-U Logical Links....132
7.6 Connectionless Data Channel....133
7.7 Operation Collision Resolution....135
7.8 Aggregating Best Effort Extended Flow Specifications ....135
7.9 Prioritizing Data over HCI....136
7.10 Supporting Extended Flow Specification for BR/EDR And BR/EDR/LE Controllers137

**8 Procedures for Flow Control and Retransmission ....139**
8.1 Information Retrieval....139
8.2 Function of PDU Types for Flow Control and Retransmission .139
8.2.1 Information frame (I-frame) ....139
8.2.2 Supervisory Frame (S-frame)....139
8.2.2.1 Receiver Ready (RR)....139
8.2.2.2 Reject (REJ) ....140
8.3 Variables and Sequence Numbers....140
8.3.1 Sending peer ....141
8.3.1.1 Send sequence number TxSeq ....141
8.3.1.2 Send state variable NextTXSeq....141
8.3.1.3 Acknowledge state variable ExpectedAckSeq....141
8.3.2 Receiving peer ....142
8.3.2.1 Receive sequence number ReqSeq ....142
8.3.2.2 Receive state variable, ExpectedTxSeq ....143
8.3.2.3 Buffer state variable BufferSeq ....143
8.4 Retransmission Mode ....144
8.4.1 Transmitting frames....144
8.4.1.1 Last received R was set to zero....144
8.4.1.2 Last received R was set to one....146
8.4.2 Receiving I-frames ....146
8.4.3 I-frames pulled by the SDU reassembly function ....146
8.4.4 Sending and receiving acknowledgements ....147
8.4.4.1 Sending acknowledgements ....147
8.4.4.2 Receiving acknowledgements ....147
8.4.5 Receiving REJ frames....148
8.4.6 Waiting acknowledgements....149

8.4.7 Exception conditions ..... 149
8.4.7.1 TxSeq Sequence error ..... 149
8.4.7.2 ReqSeq Sequence error ..... 150
8.4.7.3 Timer recovery error ..... 150
8.4.7.4 Invalid frame ..... 150
8.5 Flow Control Mode ..... 150
8.5.1 Transmitting I-frames ..... 151
8.5.2 Receiving I-frames ..... 151
8.5.3 I-frames pulled by the SDU reassembly function ..... 152
8.5.4 Sending and receiving acknowledgements ..... 152
8.5.4.1 Sending acknowledgements ..... 152
8.5.4.2 Receiving acknowledgements ..... 152
8.5.5 Waiting acknowledgements ..... 152
8.5.6 Exception conditions ..... 153
8.5.6.1 TxSeq Sequence error ..... 153
8.5.6.2 ReqSeq Sequence error ..... 154
8.5.6.3 Invalid frame ..... 154
8.6 Enhanced Retransmission Mode ..... 154
8.6.1 Function Of PDU Types ..... 155
8.6.1.1 Receiver Ready (RR) ..... 155
8.6.1.2 Reject (REJ) ..... 155
8.6.1.3 Receiver Not Ready (RNR) ..... 155
8.6.1.4 Selective Reject (SREJ) ..... 156
8.6.1.5 Functions of the Poll (P) and Final (F) bits. ..... 156
8.6.2 Rules For Timers ..... 156
8.6.2.1 Timer Rules for ACL-U Logical Links ..... 157
8.6.2.2 Timer Rules for AMP Controllers ..... 157
8.6.2.3 Timer Values used After a Move Operation ..... 159
8.6.3 General Rules for the State Machine ..... 159
8.6.4 State Diagram ..... 160
8.6.5 States Tables ..... 161
8.6.5.1 State Machines ..... 161
8.6.5.2 States ..... 161
8.6.5.3 Variables and Timers ..... 162
8.6.5.4 Events ..... 165
8.6.5.5 Conditions ..... 166
8.6.5.6 Actions ..... 168
8.6.5.7 XMIT State Table ..... 173
8.6.5.8 WAIT_F State Table ..... 173
8.6.5.9 RECV State Table[ ..... 175
8.6.5.10 REJ_SENT State Table ..... 178
8.6.5.11 SREJ_SENT State Table ..... 181
8.7 Streaming Mode ..... 184
8.7.1 Transmitting I-frames ..... 185

8.7.2 Receiving I-frames ....185
8.7.3 Exception Conditions....185
8.7.3.1 TxSeq Sequence error....185

**9 Procedure for AMP Channel Creation and Handling ....186**
9.1 Create Channel....186
9.2 Move Channel....189
9.2.1 Move Channel Protocol Procedure with Enhanced Retransmission Mode190
9.2.1.1 Enhanced Retransmission Mode Procedures During A Move Operation192
9.2.2 Move Channel Protocol Procedure with Streaming Mode (Initiator is Data Source)193
9.2.3 Move Channel Protocol Procedure with Streaming Mode (Initiator is Data Sink)194
9.3 Disconnect Channel....197

**10 List of Figures....198**

**11 List of Tables ....200**

# 1 INTRODUCTION

This section of the Bluetooth Specification defines the Logical Link Control and Adaptation Layer Protocol, referred to as L2CAP. L2CAP provides connection-oriented and connectionless data services to upper layer protocols with protocol multiplexing capability and segmentation and reassembly operation. L2CAP permits higher level protocols and applications to transmit and receive upper layer data packets (L2CAP Service Data Units, SDU) up to 64 kilobytes in length. L2CAP also permits per-channel flow control and retransmission.

The L2CAP layer provides logical channels, named L2CAP channels, which are multiplexed over one or more logical links.

## 1.1 L2CAP FEATURES

The functional requirements for L2CAP include protocol/channel multiplexing, segmentation and reassembly (SAR), per-channel flow control, and error control. L2CAP sits above a lower layer composed of one of the following:

1. BR/EDR Controller and zero or more AMP Controllers or
2. BR/EDR/LE Controller (supporting BR/EDR and LE) and zero or more AMP Controllers, or
3. LE Controller (supporting LE only)

L2CAP interfaces with upper layer protocols.

Figure 1.1 on page 31 breaks down L2CAP into its architectural components. The Channel Manager provides the control plane functionality and is responsible for all internal signaling, L2CAP peer-to-peer signaling and signaling with higher and lower layers. It performs the state machine functionality described in Section 6 on page 98 and uses message formats described in Section 4 on page 54, and Section 5 on page 79. The Retransmission and Flow Control block provides per-channel flow control and error recovery using packet retransmission. The Resource Manager is responsible for providing a frame relay service to the Channel Manager, the Retransmission and Flow Control block and those application data streams that do not require Retransmission and Flow Control services. It is responsible for coordinating the transmission and reception of packets related to multiple L2CAP channels over the facilities offered at the lower layer interface.

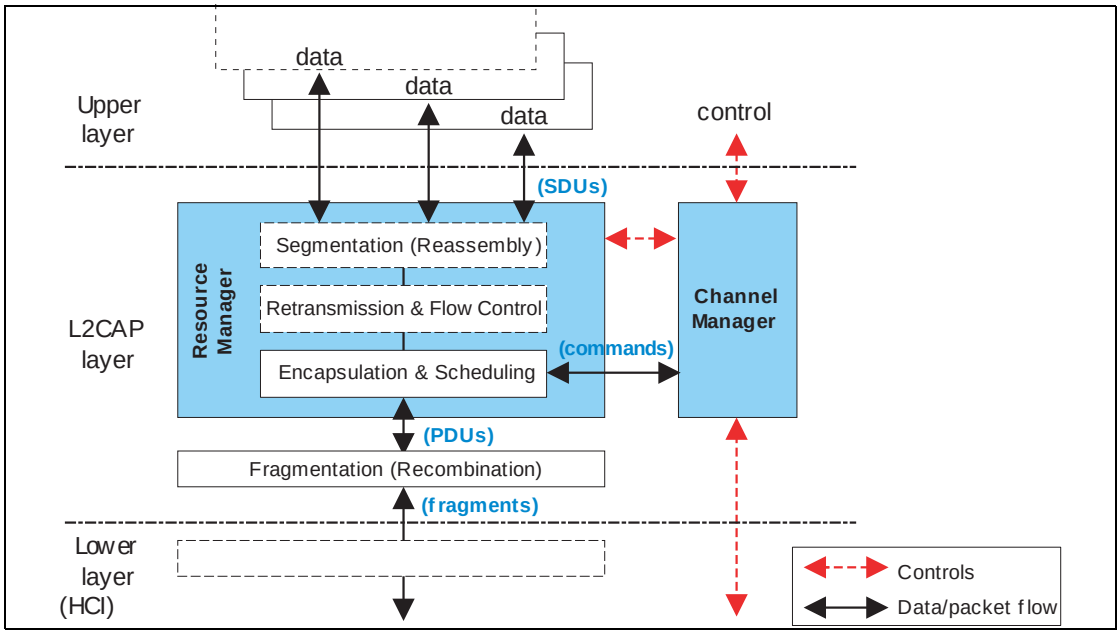



*Figure 1.1: L2CAP architectural blocks*

- *Protocol/channel multiplexing*

  L2CAP supports multiplexing over individual Controllers and across multiple controllers. An L2CAP channel shall operate over one Controller at a time.

  During channel setup, protocol multiplexing capability is used to route the connection to the correct upper layer protocol.

  For data transfer, logical channel multiplexing is needed to distinguish between multiple upper layer entities. There may be more than one upper layer entity using the same protocol.

- *Segmentation and reassembly*

  With the frame relay service offered by the Resource Manager, the length of transport frames is controlled by the individual applications running over L2CAP. Many multiplexed applications are better served if L2CAP has control over the PDU length. This provides the following benefits:

  a) Segmentation will allow the interleaving of application data units in order to satisfy latency requirements.

  b) Memory and buffer management is easier when L2CAP controls the packet size.

  c) Error correction by retransmission can be made more efficient.

  d) The amount of data that is destroyed when an L2CAP PDU is corrupted or lost can be made smaller than the application's data unit.

  e) The application is decoupled from the segmentation required to map the application packets into the lower layer packets.

- *Flow control per L2CAP channel*

Controllers provide error and flow control for data going over the air and HCI flow control exists for data going over an HCI transport. When several data streams run over the same Controller using separate L2CAP channels, each channel requires individual flow control. A window based flow control scheme is provided.

- *Error control and retransmissions*

  When L2CAP channels are moved from one Controller to another data can be lost. Also, some applications require a residual error rate much smaller than some Controllers can deliver. L2CAP provides error checks and retransmissions of L2CAP PDUs. The error checking in L2CAP protects against errors due to Controllers falsely accepting packets that contain errors but pass Controller-based integrity checks. L2CAP error checking and retransmission also protect against loss of packets due to flushing by the Controller. The error control works in conjunction with flow control in the sense that the flow control mechanism will throttle retransmissions as well as first transmissions.

- Support for Streaming

  Streaming applications such as audio set up an L2CAP channel with an agreed-upon data rate and do not want flow control mechanisms, including those in the Controller, to alter the flow of data. A flush timeout is used to keep data flowing on the transmit side. Streaming mode is used to stop HCI and Controller based flow control from being applied on the receiving side.

- *Fragmentation and Recombination*

  Some Controllers may have limited transmission capabilities and may require fragment sizes different from those created by L2CAP segmentation. Therefore layers below L2CAP may further fragment and recombine L2CAP PDUs to create fragments which fit each layer's capabilities. During transmission of an L2CAP PDU, many different levels of fragmentation and recombination may occur in both peer devices.

  The HCI driver or controller may fragment L2CAP PDUs to honor packet size constraints of a host controller interface transport scheme. This results in HCI data packet payloads carrying start and continuation fragments of the L2CAP PDU. Similarly the Controller may fragment L2CAP PDUs to map them into Controller packets. This may result in Controller packet payloads carrying start and continuation fragments of the L2CAP PDU.

  Each layer of the protocol stack may pass on different sized fragments of L2CAP PDUs, and the size of fragments created by a layer may be different in each peer device. However the PDU is fragmented within the stack, the receiving L2CAP entity still recombines the fragments to obtain the original L2CAP PDU.

- *Quality of Service*

  The L2CAP connection establishment process allows the exchange of information regarding the quality of service (QoS) expected between two Bluetooth devices. Each L2CAP implementation monitors the resources used by the protocol and ensures that QoS contracts are honored.

For a BR/EDR or BR/EDR/LE Controller, L2CAP may support both isochronous (Guaranteed) and asynchronous (Best Effort) data flows over the same ACL logical link by marking packets as automatically-flushable or non-automatically-flushable by setting the appropriate value for the Packet_Boundary_Flag in the HCI ACL Data Packet (see [vol.2, part E] Section 5.4.2 on page 428). Automatically-flushable L2CAP packets are flushed according to the automatic flush timeout set for the ACL logical link on which the L2CAP channels are mapped (see [vol.2, part E] Section 6.19 on page 444). Non-automatically-flushable L2CAP packets are not affected by the automatic flush timeout and will not be flushed. All L2CAP packets can be flushed by using the HCI Flush command (see [vol.2, part E] Section 7.3.4 on page 571).

For AMP Controllers, L2CAP places all asynchronous data flows going to the same remote device over a single logical link (aggregation). L2CAP places each isochronous data flow over its own logical link.

## 1.2 ASSUMPTIONS

The protocol is designed based on the following assumptions:

1. Controllers provide orderly delivery of data packets, although there might be individual packet corruption and duplicates. For devices with a BR/EDR or BR/EDR/LE Controller, no more than one ACL-U logical link exists between any two devices. For devices with a given AMP Controller, more than one AMP-U logical link may exist between any two devices. For devices with a BR/EDR/LE or LE Controller, no more than one LE-U logical link exists between any two devices.
2. Controllers always provide the impression of full-duplex communication channels. This does not imply that all L2CAP communications are bi-directional. Unidirectional traffic does not require duplex channels.
3. The L2CAP layer provides a channel with a degree of reliability based on the mechanisms available in Controllers and with additional packet segmentation, error detection, and retransmission that can be enabled in the enhanced L2CAP layer. Some Controllers perform data integrity checks and resend data until it has been successfully acknowledged or a timeout occurs. Other Controllers will resend data up to a certain number of times whereupon the data is flushed. Because acknowledgements may be lost, timeouts may occur even after the data has been successfully sent. Note that the use of baseband broadcast packets in a BR/EDR or BR/EDR/LE Controller is unreliable and that all broadcasts start the first segment of an L2CAP packet with the same sequence bit.
4. Controllers provide error and flow control for data going over the air and HCI flow control exists for data going over an HCI transport but some applications will want better error control than some controllers provide. Also, moving channels between Controllers requires L2CAP flow and error control. The Flow and Error Control block provides four modes. Enhanced Retransmission mode and Retransmission Mode offer segmentation, flow control

and L2CAP PDU retransmissions. Flow control mode offers just the segmentation and flow control functions. Streaming mode offers segmentation and receiver side packet flushing.

## 1.3 SCOPE

The following features are outside the scope of L2CAP's responsibilities:

- L2CAP does not transport synchronous data designated for SCO or eSCO logical transports.
- L2CAP does not support a reliable broadcast channel. See Section 3.2 on page 45.

## 1.4 TERMINOLOGY

The following formal definitions apply:

| Term | Description |
|---|---|
| Upper layer | The system layer above the L2CAP layer, which exchanges data with L2CAP in the form of SDUs. The upper layer may be represented by an application or higher protocol entity known as the Service Level Protocol. The interface of the L2CAP layer with the upper layer is not specified. |
| Lower layer | The system layer below the L2CAP layer, which exchanges data with the L2CAP layer in the form of PDUs, or fragments of PDUs. The lower layer is mainly represented within the Controller, however a Host Controller Interface (HCI) may be involved, such that an HCI host driver could also be seen as the lower layer. Except for the HCI functional specification (in case HCI is involved) the interface between L2CAP and the lower layer is not specified. |
| L2CAP channel | The logical connection between two endpoints in peer devices, characterized by their Channel Identifiers (CID), which is multiplexed over one or more Controller based logical links. |
| SDU,<br>or L2CAP SDU | Service Data Unit: a packet of data that L2CAP exchanges with the upper layer and transports transparently over an L2CAP channel using the procedures specified here. The term SDU is associated with data originating from upper layer entities only, i.e. does not include any protocol information generated by L2CAP procedures. |
| Segment,<br>or SDU segment | A part of an SDU, as resulting from the Segmentation procedure. An SDU may be split into one or more segments.<br>Note: this term is relevant only to Enhanced Retransmission mode, Streaming mode, Retransmission Mode and Flow Control Mode, not to the Basic L2CAP Mode. |

*Table 1.1: Terminology*

| Term | Description |
|---|---|
| Segmentation | A procedure used in the L2CAP Retransmission and Flow Control Modes, resulting in an SDU being split into one or more smaller units, called Segments, as appropriate for the transport over an L2CAP channel.<br>Note: this term is relevant only to the Enhanced Retransmission mode, Streaming mode, Retransmission Mode and Flow Control Mode, not to the Basic L2CAP Mode. |
| Reassembly | The reverse procedure corresponding to Segmentation, resulting in an SDU being re-established from the segments received over an L2CAP channel, for use by the upper layer. Note that the interface between the L2CAP and the upper layer is not specified; therefore, reassembly may actually occur within an upper layer entity although it is conceptually part of the L2CAP layer.<br>Note: this term is relevant only to Enhanced Retransmission mode, Streaming mode, Retransmission Mode and Flow Control Mode, not to the Basic L2CAP Mode. |
| PDU, or L2CAP PDU | Protocol Data Unit a packet of data containing L2CAP protocol information fields, control information, and/or upper layer information data. A PDU is always started by a Basic L2CAP header. Types of PDUs are: B-frames, I-frames, S-frames, C-frames and G-frames. |
| Basic L2CAP header | Minimum L2CAP protocol information that is present in the beginning of each PDU: a length field and a field containing the Channel Identifier (CID). |
| Basic information frame (B-frame) | A B-frame is a PDU used in the Basic L2CAP mode for L2CAP data packets. It contains a complete SDU as its payload, encapsulated by a Basic L2CAP header. |
| Information frame (I-frame) | An I-frame is a PDU used in Enhanced Retransmission Mode, Streaming mode, Retransmission mode, and Flow Control Mode. It contains an SDU segment and additional protocol information, encapsulated by a Basic L2CAP header |
| Supervisory frame (S-frame) | An S-frame is a PDU used in Enhanced Retransmission Mode Retransmission mode, and Flow Control Mode. It contains protocol information only, encapsulated by a Basic L2CAP header, and no SDU data. |
| Control frame (C-frame) | A C-frame is a PDU that contains L2CAP signaling messages exchanged between the peer L2CAP entities. C-frames are exclusively used on the L2CAP signaling channel. |
| Group frame (G-frame) | A G-frame is a PDU exclusively used on the Connectionless L2CAP channel. It is encapsulated by a Basic L2CAP header and contains the PSM followed by the completed SDU. G-frames may be used to broadcast data to multiple slaves (either to all slaves via Piconet Broadcast or to only active slaves via Active Broadcast) or to send unicast data to a single remote device. |

*Table 1.1: Terminology*

| Term | Description |
|---|---|
| Fragment | A part of a PDU, as resulting from a fragmentation operation. Fragments are used only in the delivery of data to and from the lower layer. They are not used for peer-to-peer transportation. A fragment may be a Start or Continuation Fragment with respect to the L2CAP PDU. A fragment does not contain any protocol information beyond the PDU; the distinction of start and continuation fragments is transported by lower layer protocol provisions.<br>Note: Start Fragments always begin with the Basic L2CAP header of a PDU. |
| Fragmentation | A procedure used to split L2CAP PDUs to smaller parts, named fragments, appropriate for delivery to the lower layer transport. Although described within the L2CAP layer, fragmentation may actually occur in an HCI host driver, and/or in a Controller, to accommodate the L2CAP PDU transport to HCI data packet or Controller packet sizes.<br>Fragmentation of PDUs may be applied in all L2CAP modes.<br>Note: in version 1.1, Fragmentation and Recombination was referred to as “Segmentation and Reassembly“. |
| Recombination | The reverse procedure corresponding to fragmentation, resulting in an L2CAP PDU re-established from fragments. In the receive path, full or partial recombination operations may occur in the Controller and/or the Host, and the location of recombination does not necessarily correspond to where fragmentations occurs on the transmit side. |
| Maximum Transmission Unit (MTU) | The maximum size of payload data, in octets, that the upper layer entity is capable of accepting, i.e. the MTU corresponds to the maximum SDU size. |
| Maximum PDU payload Size (MPS) | The maximum size of payload data in octets that the L2CAP layer entity is capable of accepting, i.e. the MPS corresponds to the maximum PDU payload size.<br>Note: in the absence of segmentation, or in the Basic L2CAP Mode, the Maximum Transmission Unit is the equivalent to the Maximum PDU payload Size and shall be made equal in the configuration parameters. |
| Signaling MTU ($MTU_{sig}$) | The maximum size of command information that the L2CAP layer entity is capable of accepting. The $MTU_{sig}$, refers to the signaling channel only and corresponds to the maximum size of a C-frame, excluding the Basic L2CAP header. The $MTU_{sig}$ value of a peer is discovered when a C-frame that is too large is rejected by the peer. |
| Connectionless MTU ($MTU_{cnl}$) | The maximum size of the connection packet information that the L2CAP layer entity is capable of accepting. The $MTU_{cnl}$ refers to the connectionless channel only and corresponds to the maximum G-frame, excluding the Basic L2CAP header and the PSM which immediately follows it. The $MTU_{cnl}$ of a peer can be discovered by sending an Information Request. |

*Table 1.1: Terminology*

| Term | Description |
|---|---|
| MaxTransmit | In Enhanced Retransmission mode and Retransmission mode, MaxTransmit controls the number of transmissions of a PDU that L2CAP is allowed to try before assuming that the PDU (and the link) is lost. The minimum value is 1 (only 1 transmission permitted). In Enhanced Retransmission mode a value 0 means infinite transmissions.<br>Note: Setting MaxTransmit to 1 prohibits PDU retransmissions. Failure of a single PDU will cause the link to drop. By comparison, in Flow Control mode, failure of a single PDU will not necessarily cause the link to drop. |

*Table 1.1: Terminology*

# 2 GENERAL OPERATION

L2CAP is based around the concept of *'channels'*. Each one of the endpoints of an L2CAP channel is referred to by a *channel identifier (CID).*

## 2.1 CHANNEL IDENTIFIERS

A channel identifier (CID) is the local name representing a logical channel endpoint on the device. The null identifier (0x0000) shall never be used as a destination endpoint. Identifiers from 0x0001 to 0x003F are reserved for specific L2CAP functions. These channels are referred to as Fixed Channels. At a minimum, the L2CAP Signaling channel (Fixed Channel 0x0001) or the L2CAP LE Signaling channel (Fixed Channel 0x0005) shall be supported. If Fixed Channel 0x0005 is supported, then Fixed Channels 0x0004 and 0x0006 shall be supported (see Table 2.1). Other fixed channels may be supported. The Information Request / Response mechanism (described in Section 4.10 and Section 4.11) shall be used to determine which fixed channels a remote device supports over the ACL-U logical link.

The characteristics of each fixed channel are defined on a per channel basis. Fixed channel characteristics include configuration parameters (e.g., reliability, MTU size, QoS), security, and the ability to change parameters using the L2CAP configuration mechanism. Table 2.1 lists the defined fixed channels, provides a reference to where the associated channel characteristics are defined and specifies the logical link over which the channel may operate. Fixed channels are available as soon as the ACL-U or LE-U logical link is set up. All initialization that is normally performed when a channel is created shall be performed for each of the supported fixed channels when the ACL-U or LE-U logical link is set up. Fixed channels shall only run over ACL-U or LE-U logical links and shall not be moved.

Implementations are free to manage the remaining CIDs in a manner best suited for that particular implementation, with the provision that two simultaneously active L2CAP channels shall not share the same CID. Table 2.1 summarizes the definition and partitioning of the CID name space.

Assignment of dynamically allocated CIDs is relative to a particular device and a device can assign CIDs independently from other devices. Thus, even if the same CID value has been assigned to (remote) channel endpoints by several remote devices connected to a single local device, the local device can still uniquely associate each remote CID with a different device.

## 2.2 OPERATION BETWEEN DEVICES

Figure 2.1 on page 40 illustrates the use of CIDs in a communication between corresponding peer L2CAP entities in separate devices. The connection-ori-

| CID | Description | Channel Characteristics | Logical Link Supported |
|---|---|---|---|
| 0x0000 | Null identifier | Not allowed | |
| 0x0001 | L2CAP Signaling channel | See Section 4 on page 54 | ACL-U |
| 0x0002 | Connectionless channel | See Section 7.6 on page 133 | ACL-U |
| 0x0003 | AMP Manager Protocol | See [Part E] Section 2.2 on page 444. | ACL-U |
| 0x0004 | Attribute Protocol | See [Vol. 3], Part F | LE-U |
| 0x0005 | Low Energy L2CAP Signaling channel | See Section 4 on page 54 | LE-U |
| 0x0006 | Security Manager Protocol | See [Vol. 3], Part H | LE-U |
| 0x0007-0x003E | Reserved | Not applicable | |
| 0x003F | AMP Test Manager | See Part D, Section 1.2.3 | ACL-U |
| 0x0040-0xFFFF | Dynamically allocated | Communicated using L2CAP configuration mechanism (see Section 7.1 on page 120. | ACL-U, AMP-U |

*Table 2.1: CID name space*

ented data channels represent a connection between two devices, where a CID identifies each endpoint of the channel. When used for broadcast transmissions, the connectionless channel restricts data flow to a single direction. The connectionless channel may be used to transmit data to all slaves in a piconet (utilizing Piconet Broadcast which includes both active and parked slaves) or to all active slaves (using Active Broadcast which includes only active slaves). When used for unicast transmissions the connectionless channel may be used in either direction between a master and a slave.

There are also a number of CIDs reserved for special purposes. The L2CAP signaling channel is one example of a reserved channel. This channel is used to create and establish connection-oriented data channels and to negotiate changes in the characteristics of connection-oriented channels and to discover characteristics of the connectionless channel operating over the ACL-U logical link.

The L2CAP signaling channel and all supported fixed channels are available immediately when the ACL-U logical link is established between two devices. Another CID (0x0002) is reserved for all incoming and outgoing connectionless

data traffic, whether broadcast or unicast. Connectionless data traffic may flow immediately once the ACL-U logical link is established between two devices and once the transmitting device has determined that the remote device supports connectionless traffic.

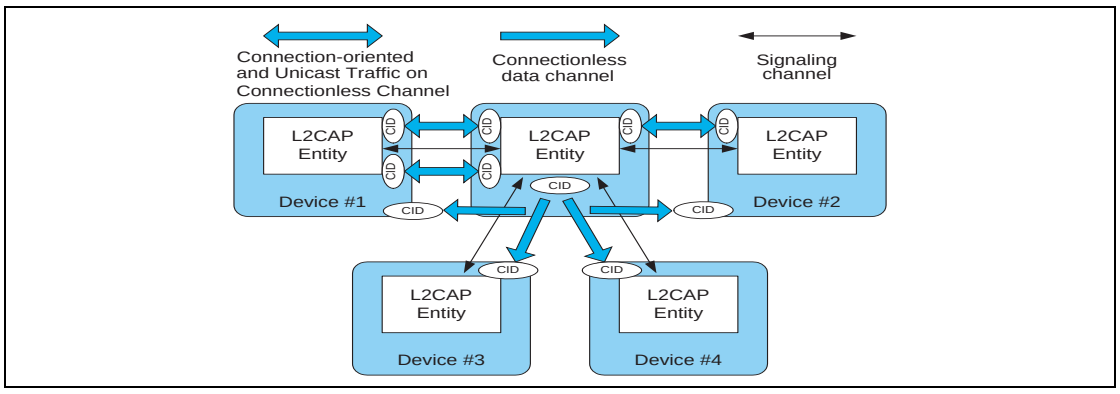



*Figure 2.1: Channels between devices*

Table 2.2 describes the various channel types and their source and destination identifiers. A dynamically allocated CID is allocated to identify the local endpoint and shall be in the range 0x0040 to 0xFFFF. Section 6 on page 98 describes the state machine associated with each connection-oriented channel with a dynamically allocated CID. Section 3.1 on page 44 and Section 3.3 on page 45 describe the packet format associated with connection-oriented channels. Section 3.2 on page 45 describes the packet format associated with the connectionless channel.

| Channel Type | Local CID (sending) | Remote CID (receiving) |
|---|---|---|
| Connection-oriented | Dynamically allocated and fixed | Dynamically allocated and fixed |
| Connectionless data | 0x0002 (fixed) | 0x0002 (fixed) |
| L2CAP Signaling | 0x0001 and 0x0005 (fixed) | 0x0001 and 0x0005 (fixed) |

*Table 2.2: Types of Channel Identifiers*

## 2.3 OPERATION BETWEEN LAYERS

L2CAP implementations should follow the general architecture described below. L2CAP implementations transfer data between upper layer protocols and the lower layer protocol. This document lists a number of services that should be exported by any L2CAP implementation. Each implementation shall also support a set of signaling commands for use between L2CAP implementations. L2CAP implementations should also be prepared to accept certain types of events from lower layers and generate events to upper layers. How these events are passed between layers is implementation specific.

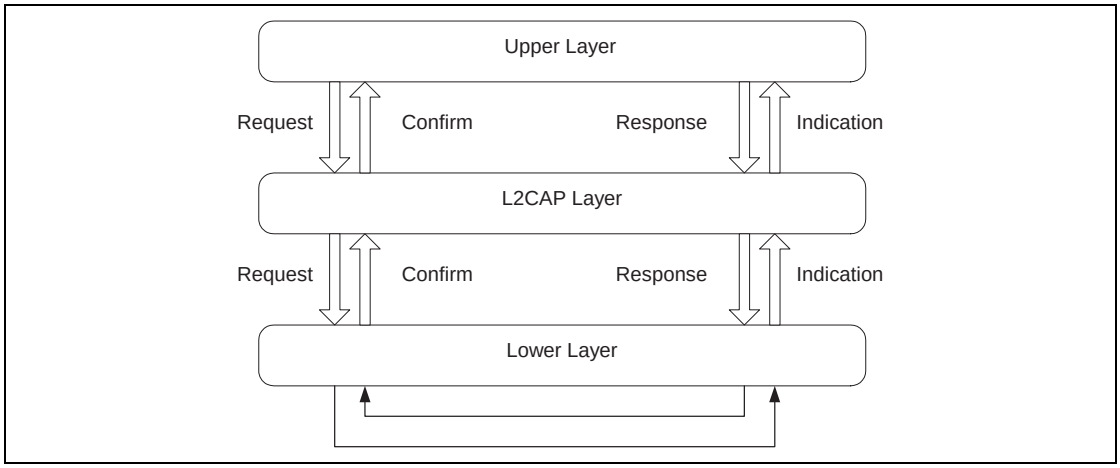



*Figure 2.2: L2CAP transaction model*

## 2.4 MODES OF OPERATION

L2CAP may operate in one of five different modes as selected for each L2CAP channel.

The modes are:

- Basic L2CAP Mode (equivalent to L2CAP specification in Bluetooth v1.1) [1]
- Flow Control Mode
- Retransmission Mode
- Enhanced Retransmission Mode
- Streaming Mode

The modes are enabled using the configuration procedure described in Section 7.1. The Basic L2CAP Mode shall be the default mode, which is used when no other mode is agreed. Enhanced Retransmission mode shall be used for all reliable channels created over AMP-U logical links and for ACL-U logical links operating as described in Section 7.10. Enhanced Retransmission mode should be enabled for reliable channels created over ACL-U logical links not operating as described in Section 7.10. Streaming mode shall be used for streaming applications created over AMP-U logical links and -ACL-U logical links operating as described in Section 7.10. Streaming mode should be enabled for streaming applications created over ACL-U logical links not operating as described in Section 7.10. Either Enhanced Retransmission mode or Streaming mode should be enabled when supported by both L2CAP entities. Flow Control Mode and Retransmission mode shall only be enabled when communicating with L2CAP entities that do not support either Enhanced Retransmission mode or Streaming mode.

1. Specification of the Bluetooth System v1.1 (Feb 22nd 2001): volume 1, part D.

In Flow Control mode, Retransmission mode, and Enhanced Retransmission mode, PDUs exchanged with a peer entity are numbered and acknowledged. The sequence numbers in the PDUs are used to control buffering, and a TxWindow size is used to limit the required buffer space and/or provide a method for flow control.

In Flow Control Mode no retransmissions take place, but missing PDUs are detected and can be reported as lost.

In Retransmission Mode a timer is used to ensure that all PDUs are delivered to the peer, by retransmitting PDUs as needed. A go-back-n repeat mechanism is used to simplify the protocol and limit the buffering requirements.

Enhanced Retransmission mode is similar to Retransmission mode. It adds the ability to set a POLL bit to solicit a response from a remote L2CAP entity, adds the SREJ S-frame to improve the efficiency of error recovery and adds an RNR S-frame to replace the R-bit for reporting a local busy condition.

Streaming mode is for real-time isochronous traffic. PDUs are numbered but are not acknowledged. A finite flush timeout is set on the sending side to flush packets that are not sent in a timely manner. On the receiving side if the receive buffers are full when a new PDU is received then a previously received PDU is overwritten by the newly received PDU. Missing PDUs can be detected and reported as lost. TxWindow size is not used in Streaming mode.

**Note**: Although L2CAP Basic mode may be used for L2CAP channels operating over ACL-U logical links, only L2CAP channels which have been configured to use Enhanced Retransmission mode or Streaming mode may be moved to operate over an AMP-U logical link. (see Section 4.16).

L2CAP channels used for multiplexing layers such as RFCOMM can serve a variety of higher layer protocols. In cases where the local device and remote device have mutual support for an AMP and complementary support for profiles on a multiplexing layer which requires reliability (e.g. RFCOMM), the multiplexing layer should be configured to use Enhanced Retransmission mode to ensure that profiles utilizing the multiplexer are not prevented from being moved to the AMP-U logical link. Similarly, in cases where the local device and remote device have mutual support for an AMP and complementary support for profiles on a multiplexing layer which does not require reliability, the multiplexing layer should be configured to use Streaming mode to ensure that profiles utilizing the multiplexer are not prevented from being moved to the AMP-U logical link.

Care should be taken in selecting the parameters used for Enhanced Retransmission mode and Streaming mode when they are used beneath legacy profile implementations to ensure that performance is not negatively impacted relative to the performance achieved when using the same profile with Basic mode on a ACL-U logical link. When there can never be a mutually supported AMP, nor complementary support for a profile which would benefit from AMP, it may be

preferable to configure Basic mode to minimize the risk of negative performance impacts.

## 2.5 MAPPING CHANNELS TO LOGICAL LINKS

L2CAP maps channels to Controller logical links, which in turn run over Controller physical links. All logical links going between a local Controller and remote Controller run over a single physical link. There is one ACL-U logical link per BR/EDR physical link and one LE-U logical link per LE physical link, while there may be multiple AMP-U logical links per AMP physical link.

All Best Effort and Guaranteed channels going over a BR/EDR physical link between two devices shall be mapped to a single ACL-U logical link. All Best Effort channels going over an AMP physical link between two Controllers shall be mapped to a single AMP-U logical link while each Guaranteed channel going between two Controllers shall be mapped to its own AMP-U logical link—one AMP-U logical link per Guaranteed channel. All channels going over an LE physical link between two devices shall be treated as best effort and mapped to a single LE-U logical link.

When a Guaranteed channel is created or moved to a Controller, a corresponding Guaranteed logical link shall be created to carry the channel traffic. Creation of a Guaranteed logical link involves admission control. Admission control is verifying that the guarantee can be achieved without compromising existing guarantees. For an AMP Controller, L2CAP shall tell the controller to create a Guaranteed logical link and admission control shall be performed by the Controller. For a BR/EDR Controller, admission control (creation of a Guaranteed logical link) shall be performed by the L2CAP layer.

# 3 DATA PACKET FORMAT

L2CAP is packet-based but follows a communication model based on *channels*. A channel represents a data flow between L2CAP entities in remote devices. Channels may be connection-oriented or connectionless. Fixed channels other than the L2CAP connectionless channel (CID 0x0002) and the two L2CAP signaling channels (CIDs 0x0001 and 0x0005) are considered connection-oriented. All channels with dynamically assigned CIDs are connection-oriented. All L2CAP layer packet fields shall use Little Endian byte order with the exception of the information payload field. The endian-ness of higher layer protocols encapsulated within L2CAP information payload is protocol-specific.

## 3.1 CONNECTION-ORIENTED CHANNELS IN BASIC L2CAP MODE

Figure 3.1 on page 44 illustrates the format of the L2CAP PDU used on connection-oriented channels. In basic L2CAP mode, the L2CAP PDU on a connection-oriented channel is also referred to as a "B-frame."

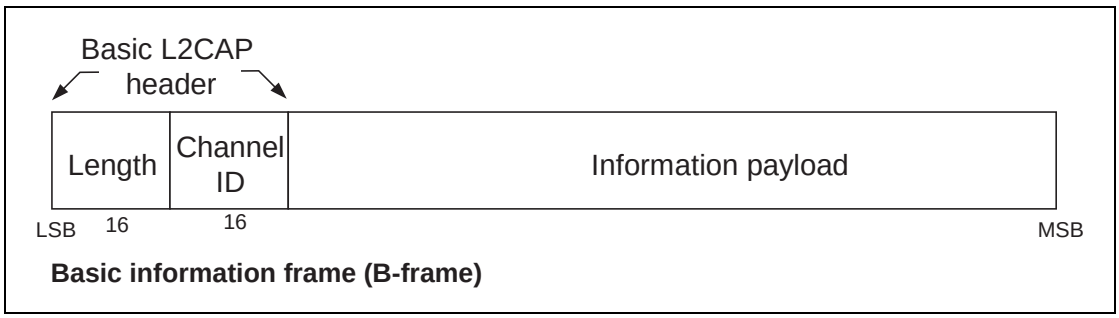



*Figure 3.1: L2CAP PDU format in Basic L2CAP mode on connection-oriented channels (field sizes in bits)*

The fields shown are:

- *Length: 2 octets (16 bits)*

  Length indicates the size of the information payload in octets, excluding the length of the L2CAP header. The length of an information payload can be up to 65535 octets. The Length field is used for recombination and serves as a simple integrity check of the recombined L2CAP packet on the receiving end.

- *Channel ID: 2 octets*

  The channel ID (CID) identifies the destination channel endpoint of the packet.

- *Information payload: 0 to 65535 octets*

  This contains the payload received from the upper layer protocol (outgoing packet), or delivered to the upper layer protocol (incoming packet). The MTU for channels with dynamically allocated CIDs is determined during channel

configuration (see Section 5.1 on page 79). The minimum supported MTU values for the signaling PDUs are shown in Table 4.1 on page 54).

## 3.2 CONNECTIONLESS DATA CHANNEL IN BASIC L2CAP MODE

Figure 3.2 illustrates the L2CAP PDU format within a connectionless data channel. Here, the L2CAP PDU is also referred to as a "G-frame."

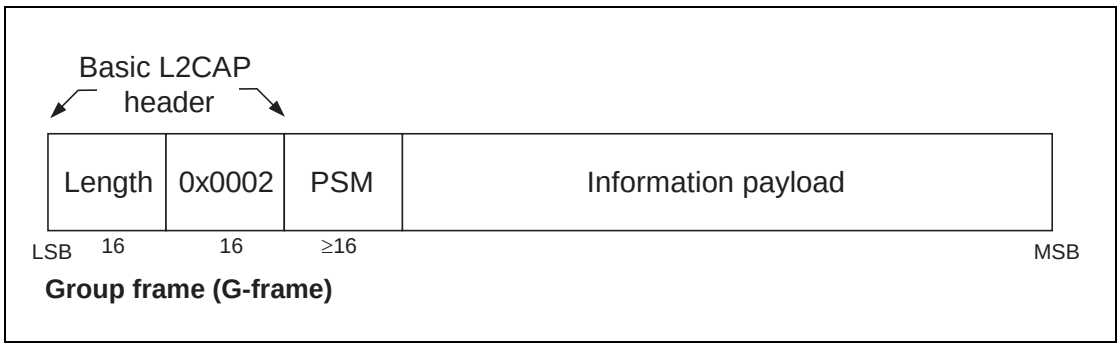



*Figure 3.2: L2CAP PDU format on the Connectionless channel*

The fields shown are:

- *Length: 2 octets*

  Length indicates the size of information payload plus the PSM field in octets.

- *Channel ID: 2 octets*

  Channel ID (0x0002) reserved for connectionless traffic.

- *Protocol/Service Multiplexer (PSM): 2 octets (minimum)*

  For information on the PSM field see Section 4.2 on page 58.

- *Information payload: 0 to 65533 octets*

  This parameter contains the payload information to be distributed to all slaves in the piconet for broadcast connectionless traffic, or to a specific remote device for data sent via the L2CAP connectionless channel. Implementations shall support a connectionless MTU ($MTU_{cnl}$) of 48 octets on the connectionless channel. Devices may also explicitly change to a larger or smaller connectionless MTU ($MTU_{cnl}$).

  Note: the maximum size of the Information payload field decreases accordingly if the PSM field is extended beyond the two octet minimum.

## 3.3 CONNECTION-ORIENTED CHANNEL IN RETRANSMISSION/FLOW CONTROL/STREAMING MODES

To support flow control, retransmissions, and streaming, L2CAP PDU types with protocol elements in addition to the Basic L2CAP header are defined. The information frames (I-frames) are used for information transfer between L2CAP

entities. The supervisory frames (S-frames) are used to acknowledge I-frames and request retransmission of I-frames.

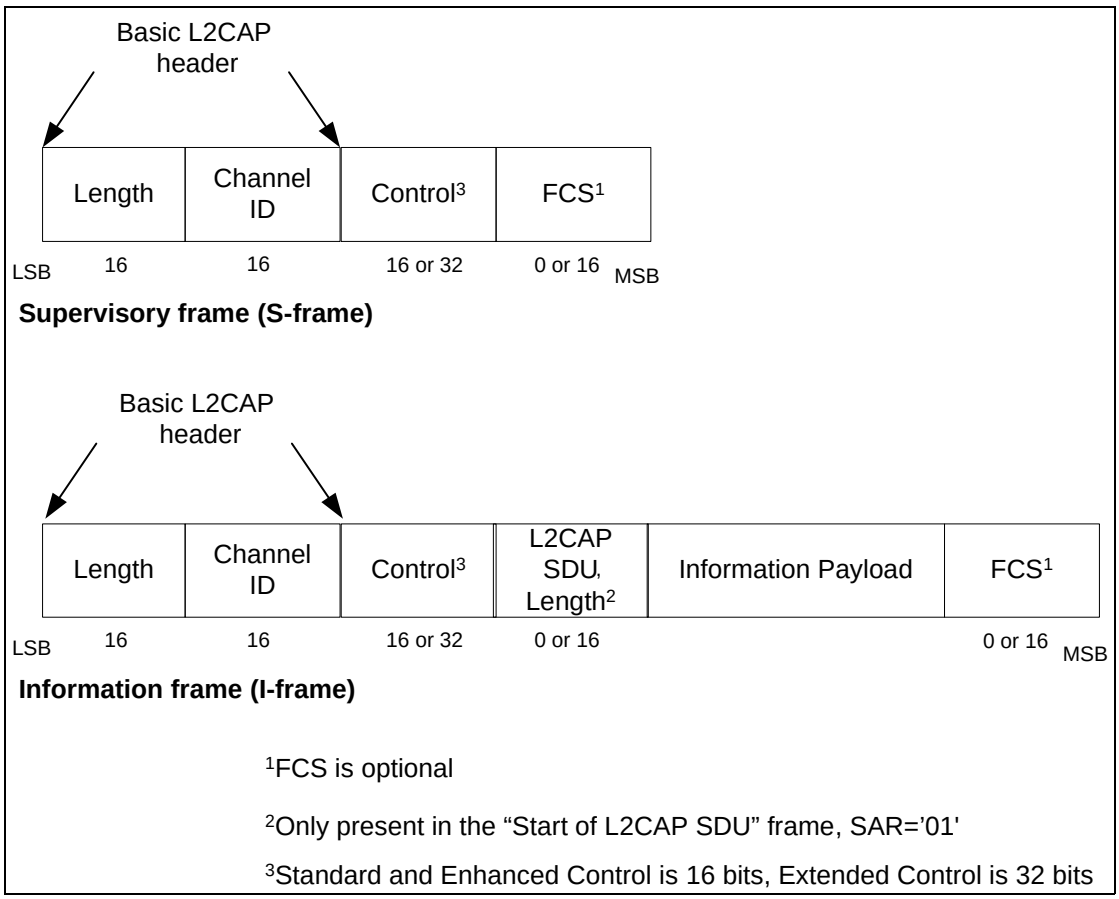



*Figure 3.3: L2CAP PDU formats in Flow Control and Retransmission Modes*

### 3.3.1 L2CAP header fields

- *Length: 2 octets*

  The first two octets in the L2CAP PDU contain the length of the entire L2CAP PDU in octets, excluding the Length and CID field.

  For I-frames and S-frames, the Length field therefore includes the octet lengths of the Control, L2CAP SDU Length (when present), Information octets and frame check sequence (FCS) (when present) fields.

  The maximum number of Information octets in one I-frame is based on which fields are present and the type of the Control Field. The maximum number of Information octets in an I-frame with a Standard Control field is as follows:

| | |
|---|---|
| L2CAP SDU Length present and FCS present | 65529 octets |
| L2CAP SDU Length present and FCS not present | 65531 octets |
| L2CAP SDU Length not present and FCS present | 65531 octets |
| L2CAP SDU Length not present and FCS not present | 65533 octets |

The maximum number of Information octets in an I-frame with an Extended Control field is as follows:

| | |
|---|---|
| L2CAP SDU Length present and FCS present | 65527 octets |
| L2CAP SDU Length present and FCS not present | 65529 octets |
| L2CAP SDU Length not present and FCS present | 65529 octets |
| L2CAP SDU Length not present and FCS not present | 65531 octets |

- *Channel ID: 2 octets*

  This field contains the Channel Identification (CID).

### 3.3.2 Control field (2 or 4 octets)

The Control Field identifies the type of frame. There are three different Control Field formats: the Standard Control Field, the Enhanced Control Field, and the Extended Control Field. The Standard Control Field shall be used for Retransmission mode and Flow Control mode. The Enhanced Control Field shall be used for Enhanced Retransmission mode and Streaming mode. The Extended Control Field may be used for Enhanced Retransmission mode and Streaming mode. The Control Field will contain sequence numbers where applicable. Its coding is shown in Table 3.1 on page 48, Table 3.2 on page 49, and Table 3.3 on page 49. There are two different frame types, Information frame types and Supervisory frame types. Information and Supervisory frames types are distinguished by the least significant bit in the Control Field, as shown in Table 3.1, Table 3.2 on page 49, and Table 3.3 on page 49.

- *Information frame format (I-frame)*

  The I-frames are used to transfer information between L2CAP entities. Each I-frame has a TxSeq(Send sequence number), ReqSeq(Receive sequence number) which may or may not acknowledge additional I-frames received by the data link layer entity. Each I-frame with a Standard Control field has a retransmission bit (R bit) that affects whether I-frames are retransmitted. Each I-frame with an Enhanced Control Field or an Extended Control Field has an F-bit that is used in Poll/Final bit functions.

  The SAR field in the I-frame is used for segmentation and reassembly control. The L2CAP SDU Length field specifies the length of an SDU, including the aggregate length across all segments if segmented.

- *Supervisory frame format (S-frame)*

  S-frames are used to acknowledge I-frames and request retransmission of I-frames. Each S-frame has an ReqSeq sequence number which may acknowledge additional I-frames received by the data link layer entity. Each S-frame with a Standard Control Field has a retransmission bit (R bit) that affects whether I-frames are retransmitted. Each S-frame with an Enhanced Control field or an Extended Control Field has a Poll bit (P-bit) and a Final bit (F-bit) and does not have an R-bit.

  Defined types of S-frames are RR (Receiver Ready), REJ (Reject), RNR (Receiver Not Ready) and SREJ (Selective Reject).

| Frame type | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| I | SAR | | ReqSeq | | | | | | R | TxSeq | | | | | | 0 |
| S | X | X | ReqSeq | | | | | | R | X | X | X | S | | 0 | 1 |

X denotes reserved bits, which shall be set to 0.

*Table 3.1: Standard Control Field formats*

| Frame type | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| I | SAR | | ReqSeq | | | | | | F | TxSeq | | | | | | 0 |
| S | X | X | ReqSeq | | | | | | F | X | X | P | S | | 0 | 1 |

X denotes reserved bits, which shall be set to 0.

*Table 3.2: Enhanced Control Field formats*

| Frame type | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| I | ReqSeq | | | | | | | | | | | | | | F | 0 |
| | TxSeq | | | | | | | | | | | | | | SAR | |
| S | ReqSeq | | | | | | | | | | | | | | F | 1 |
| | | | | | | | | | X | X | X | X | X | P | S | |

X denotes reserved bits, which shall be set to 0.

*Table 3.3: Extended Control Field formats*

- *Send Sequence Number - TxSeq (6 bits or 14 bits)*

  The send sequence number is used to number each I-frame, to enable sequencing and retransmission.

- *Receive Sequence Number - ReqSeq (6 bits or 14 bits)*

  The receive sequence number is used by the receiver side to acknowledge I-frames, and in the REJ and SREJ frames to request the retransmission of an I-frame with a specific send sequence number.

- *Retransmission Disable Bit - R (1 bit)*

  The Retransmission Disable bit is used to implement Flow Control. The receiver sets the bit when its internal receive buffer is full, this happens when one or more I-frames have been received but the SDU reassembly function has not yet pulled all the frames received. When the sender receives a frame with the Retransmission Disable bit set it shall disable the RetransmissionTimer, this causes the sender to stop retransmitting I-frames.

  R=0: Normal operation. Sender uses the RetransmissionTimer to control retransmission of I-frames. Sender does not use the MonitorTimer.

  R=1: Receiver side requests sender to postpone retransmission of I-frames. Sender monitors signaling with the MonitorTimer. Sender does not use the RetransmissionTimer.

  The functions of ReqSeq and R are independent.

- *Segmentation and Reassembly - SAR (2 bits)*

The SAR bits define whether an L2CAP SDU is segmented. For segmented SDUs, the SAR bits also define which part of an SDU is in this I-frame, thus allowing one L2CAP SDU to span several I-frames.

An I-frame with SAR=”Start of L2CAP SDU” also contains a length indicator, specifying the number of information octets in the total L2CAP SDU. The encoding of the SAR bits is shown in Table 3.4.

| | |
|---|---|
| 00 | Unsegmented L2CAP SDU |
| 01 | Start of L2CAP SDU |
| 10 | End of L2CAP SDU |
| 11 | Continuation of L2CAP SDU |

*Table 3.4: SAR control element format.*

- *Supervisory function - S (2 bits)*

  The S-bits mark the type of S-frame. There are four types defined: RR (Receiver Ready), REJ (Reject), RNR (Receiver Not Ready) and SREJ (Selective Reject). The encoding is shown in Table 3.5.

| | |
|---|---|
| 00 | RR - Receiver Ready |
| 01 | REJ - Reject |
| 10 | RNR - Receiver Not Ready |
| 11 | SREJ - Select Reject |

*Table 3.5: S control element format: type of S-frame.*

- *Poll - P (1 bit)*

  The P-bit is set to 1 to solicit a response from the receiver. The receiver shall respond immediately with a frame with the F-bit set to 1.

- *Final - F (1 bit)*

  The F-bit is set to 1 in response to an S-frame with the P bit set to 1.

### 3.3.3 L2CAP SDU Length Field (2 octets)

When a SDU spans more than one I-frame, the first I-frame in the sequence shall be identified by SAR=01=”Start of L2CAP SDU”. The L2CAP SDU Length field shall specify the total number of octets in the SDU. The L2CAP SDU Length field shall be present in I-frames with SAR=01 (Start of L2CAP SDU), and shall not be used in any other I-frames. When the SDU is unsegmented (SAR=00), L2CAP SDU Length field is not needed and shall not be present.

### 3.3.4 Information Payload Field

The information payload field consists of an integral number of octets. The maximum number of octets in this field is the same as the negotiated value of

the MPS configuration parameter. The maximum number of octets in this field is also limited by the range of the Basic L2CAP header length field. This ranges from 65533 octets for I-frames with a Standard or Enhanced Control Field, no SDU length field, and no FCS field to 65527 octets for I-frames with an Enhanced Control Field, an SDU length field, and FCS field. Thus, even if an MPS of 65533 has been negotiated, the range of the Basic L2CAP header length field will restrict the number of octets in this field. For example, when an Enhanced Control Field, an SDU length field, and FCS field are present the number of octets in this field is restricted to 65529.

### 3.3.5 Frame Check Sequence (2 octets)

The Frame Check Sequence (FCS) is 2 octets. The FCS is constructed using the generator polynomial $g(D) = D^{16} + D^{15} + D^2 + 1$ (see Figure 3.4). The 16 bit LFSR is initially loaded with the value 0x0000, as depicted in Figure 3.5. The switch S is set in position 1 while data is shifted in, LSB first for each octet. After the last bit has entered the LFSR, the switch is set in position 2, and the register contents are transmitted from right to left (i.e. starting with position 15, then position 14, etc.). The FCS covers the Basic L2CAP header, Control, L2CAP-SDU length and Information payload fields, if present, as shown in Figure 3.3 on page 46.

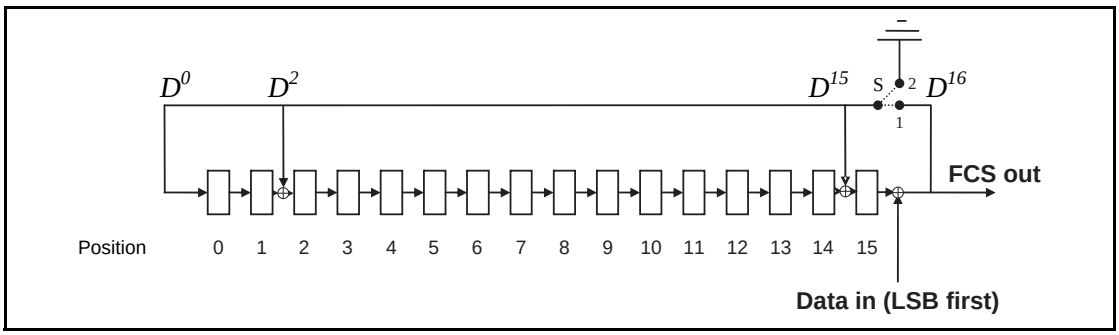



*Figure 3.4: The LFSR circuit generating the FCS.*

| Position | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| LFSR | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

*Figure 3.5: Initial state of the FCS generating circuit.*

Examples of FCS calculation, $g(D) = D^{16} + D^{15} + D^2 + 1$:

1. ***I Frame***
   Length = 14
   Control = 0x0002 (SAR=0, ReqSeq=0, R=0, TxSeq=1)
   Information Payload = 00 01 02 03 04 05 06 07 08 09 (10 octets, hexadecimal notation)

==> FCS = 0x6138
==> Data to Send = 0E 00 40 00 02 00 00 01 02 03 04 05 06 07 08 09 38 61 (hexadecimal notation)

2. ***RR Frame***
   Length = 4
   Control = 0x0101 (ReqSeq=1, R=0, S=0)
   ==> FCS = 0x14D4
   ==> Data to Send = 04 00 40 00 01 01 D4 14 (hexadecimal notation)

### 3.3.6 Invalid Frame Detection

For Retransmission mode and Flow Control mode, a received PDU shall be regarded as invalid, if one of the following conditions occurs:

1. Contains an unknown CID.
2. Contains an FCS error.
3. Contains a length greater than the maximum PDU payload size (MPS).
4. I-frame that has fewer than 8 octets.
5. I-frame with SAR=01 (Start of L2CAP SDU) that has fewer than 10 octets.
6. I-frame with SAR bits that do not correspond to a normal sequence of either unsegmented or start, continuation, end for the given CID.
7. S-frame where the length field is not equal to 4.

These error conditions may be used for error reporting.

### 3.3.7 Invalid Frame Detection Algorithm

For Enhanced Retransmission mode and Streaming mode the following algorithm shall be used for received PDUs. It may be used for Retransmission mode and Flow Control mode:

1. Check the CID. If the PDU contains an unknown CID then it shall be ignored.
2. Check the FCS. If the PDU contains an FCS error then it shall be ignored. If the channel is configured to use "No FCS" then the PDU is considered to have a good FCS (no FCS error).
3. Check the following conditions. If one of the conditions occurs the channel shall be closed or in the case of fixed channels the ACL shall be disconnected.
   a) PDU contains a length greater than the maximum PDU payload size (MPS)
   b) I-frame that has fewer than the required number of octets.If the channel is configured to use a Standard or Enhanced Control Field then the required number of octets is 6 if "No FCS" is configured; otherwise, it is 8. If the channel is configured to use the Extended

Control Field then the required number of octets is 8 if "No FCS" is configured; otherwise, it is 10.

c) S-frame where the length field is invalid. If the channel is configured to use a Standard or Enhanced Control Field then the length field shall be 2 if "No FCS" is configured; otherwise, the length field shall be 4. If the channel is configured to use the Extended Control Field then the length field shall be 4 if "No FCS" is configured; otherwise, the length field shall be 6.

4. Check the SAR bits. The SAR check is performed after the frame has been successfully received in the correct sequence. The PDU is invalid if one of the following conditions occurs:

   a) I-frame with SAR=01 (Start of L2CAP SDU) that has fewer than the required number of octets. If the channel is configured to use a Standard or Enhanced Control field then the required number of octets is 8 if "No FCS" is configured; otherwise, the required number of octets is 10. If the channel is configured to use an Extended Control field then the required number of octets is 10 if "No FCS" is configured; otherwise, the required number of octets is 12.

   b) I-frame with SAR bits that do not correspond to a normal sequence of either unsegmented or start, continuation, end for the given CID.

   c) I-frame with SAR= 01 (Start of L2CAP SDU) where the value in the L2CAP SDU length field exceeds the configured MTU.

5. If the I-frame has been received in the correct sequence and is invalid as described in 4 then the channel shall be closed or in the case of fixed channels the ACL shall be disconnected. For Streaming mode and Flow Control mode if one or more I-frames are missing from a sequence of I-frames using SAR bits of start, continuation and end then received I-frames in the sequence may be ignored. For Flow Control mode and Streaming mode I-frames received out of sequence with SAR bits of unsegmented may be accepted.

If the algorithm is used for Retransmission mode or Flow control mode then it shall be used instead of Invalid Frame detection described in section 3.3.6.

These error conditions may be used for error reporting.

# 4 SIGNALING PACKET FORMATS

This section describes the signaling commands passed between two L2CAP entities on peer devices. All signaling commands are sent over a signaling channel. The signaling channel for managing channels over ACL-U logical links shall use CID 0x0001 and the signaling channel for managing channels over LE-U logical links shall use CID 0x0005. Signaling channels are available as soon as the lower layer logical transport is set up and L2CAP traffic is enabled. Figure 4.1 on page 54 illustrates the general format of L2CAP PDUs containing signaling commands (C-frames). Multiple commands may be sent in a single C-frame over Fixed Channel CID 0x0001 while only one command per C-frame shall be sent over Fixed Channel CID 0x0005. Commands take the form of Requests and Responses. All L2CAP implementations shall support the reception of C-frames with a payload length that does not exceed the signaling MTU. The minimum supported payload length for the C-frame ($MTU_{sig}$) is defined in Table 4.1 on page 54. L2CAP implementations should not use C-frames that exceed the $MTU_{sig}$ of the peer device. If a device receives a C-frame that exceeds its $MTU_{sig}$ then it shall send a Command Reject containing the supported $MTU_{sig}$. Implementations shall be able to handle the reception of multiple commands in an L2CAP packet sent over Fixed Channel CID 0x0001.

| Logical Link | Minimum Supported Payload Length for the C-frame ($MTU_{sig}$) |
|---|---|
| ACL-U not supporting Extended Flow Specification | 48 octets |
| ACL-U supporting the Extended Flow Specification feature | 672 octets |
| LE-U | 23 octets |

*Table 4.1: Minimum Signaling MTU*

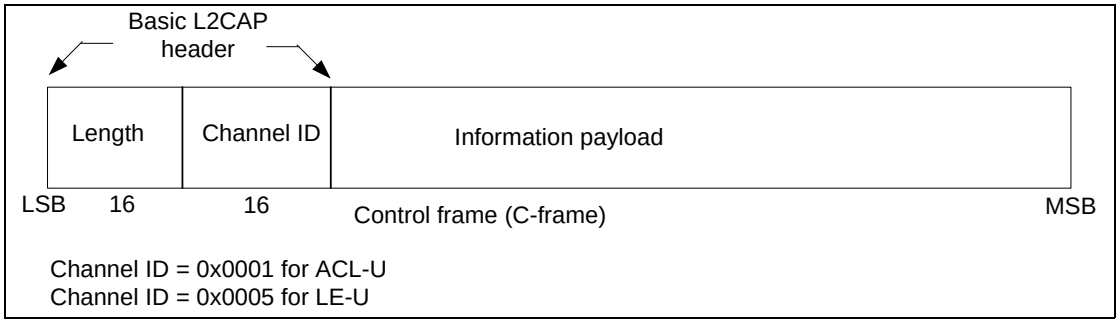



*Figure 4.1: L2CAP PDU format on a signaling channel*

Figure 4.2 displays the general format of all signaling commands.

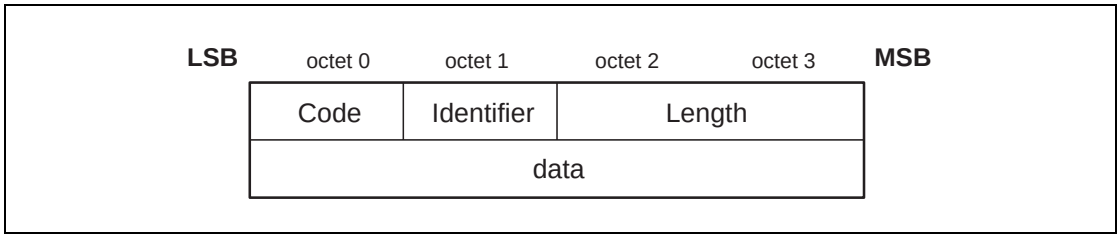



*Figure 4.2: Command format*

The fields shown are:

- *Code (1 octet)*

  The Code field is one octet long and identifies the type of command. When a packet is received with a Code field that is unknown or disallowed on the signaling channel it is received on, a Command Reject packet (defined in Section 4.1 on page 56) is sent in response.

  Table 4.2 on page 55 lists the codes defined by this document. All codes are specified with the most significant bit in the left-most position.

| Code | Description | CIDs on which Code is Allowed |
|---|---|---|
| 0x00 | Reserved | Any |
| 0x01 | Command reject | 0x0001 and 0x0005 |
| 0x02 | Connection request | 0x0001 |
| 0x03 | Connection response | 0x0001 |
| 0x04 | Configure request | 0x0001 |
| 0x05 | Configure response | 0x0001 |
| 0x06 | Disconnection request | 0x0001 |
| 0x07 | Disconnection response | 0x0001 |
| 0x08 | Echo request | 0x0001 |
| 0x09 | Echo response | 0x0001 |
| 0x0A | Information request | 0x0001 |
| 0x0B | Information response | 0x0001 |
| 0x0C | Create Channel request | 0x0001 |
| 0x0D | Create Channel response | 0x0001 |
| 0x0E | Move Channel request | 0x0001 |
| 0x0F | Move Channel response | 0x0001 |
| 0x10 | Move Channel Confirmation | 0x0001 |
| 0x11 | Move Channel Confirmation response | 0x0001 |

*Table 4.2: Signaling Command Codes*

| Code | Description | CIDs on which Code is Allowed |
|---|---|---|
| 0x12 | Connection Parameter Update request | 0x0005 |
| 0x13 | Connection Parameter Update response | 0x0005 |

*Table 4.2: Signaling Command Codes*

- *Identifier (1 octet)*

  The Identifier field is one octet long and matches responses with requests. The requesting device sets this field and the responding device uses the same value in its response. Between any two devices a different Identifier shall be used for each successive command. Following the original transmission of an Identifier in a command, the Identifier may be recycled if all other Identifiers have subsequently been used.

  RTX and ERTX timers are used to determine when the remote end point is not responding to signaling requests. On the expiration of a RTX or ERTX timer, the same identifier shall be used if a duplicate Request is re-sent as stated in Section 6.1.7 on page 110.

  A device receiving a duplicate request should reply with a duplicate response. A command response with an invalid identifier is silently discarded. Signaling identifier 0x00 is an illegal identifier and shall never be used in any command.

- *Length (2 octets)*

  The Length field is two octets long and indicates the size in octets of the data field of the command only, i.e., it does not cover the Code, Identifier, and Length fields.

- *Data (0 or more octets)*

  The Data field is variable in length. The Code field determines the format of the Data field. The length field determines the length of the data field.

## 4.1 COMMAND REJECT (CODE 0x01)

A Command Reject packet shall be sent in response to a command packet with an unknown command code or when sending the corresponding response is inappropriate. Figure 4.3 on page 57 displays the format of the packet. The identifier shall match the identifier of the command packet being rejected. Implementations shall always send these packets in response to unidentified signaling packets. Command Reject packets should not be sent in response to an identified Response packet.

When multiple commands are included in an L2CAP packet and the packet exceeds the signaling MTU ($MTU_{sig}$) of the receiver, a single Command Reject packet shall be sent in response. The identifier shall match the first Request command in the L2CAP packet. If only Responses are recognized, the packet shall be silently discarded.

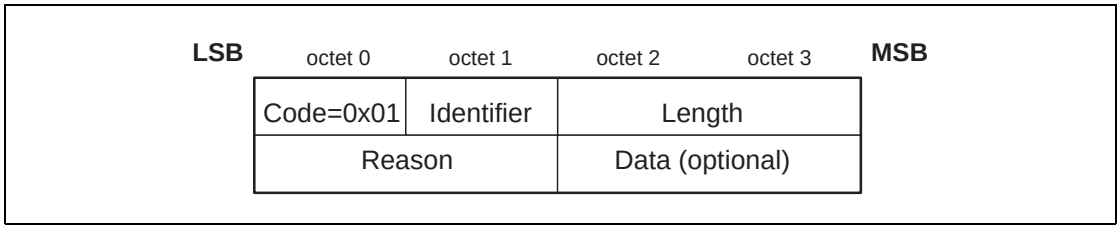



*Figure 4.3: Command Reject packet*

Figure 4.3 shows the format of the Command Reject packet. The data fields are:

- *Reason (2 octets)*

  The Reason field describes why the Request packet was rejected, and is set to one of the Reason codes in Table 4.3.

| Reason value | Description |
|---|---|
| 0x0000 | Command not understood |
| 0x0001 | Signaling MTU exceeded |
| 0x0002 | Invalid CID in request |
| Other | Reserved |

*Table 4.3: Reason Code Descriptions*

- *Data (0 or more octets)*

  The length and content of the Data field depends on the Reason code. If the Reason code is 0x0000, “Command not understood”, no Data field is used. If the Reason code is 0x0001, “Signaling MTU Exceeded”, the 2-octet Data field represents the maximum signaling MTU the sender of this packet can accept.

  If a command refers to an invalid channel then the Reason code 0x0002 will be returned. Typically a channel is invalid because it does not exist. The data field shall be 4 octets containing the local (first) and remote (second) channel endpoints (relative to the sender of the Command Reject) of the disputed channel. The remote endpoint is the source CID from the rejected command. The local endpoint is the destination CID from the rejected command. If the rejected command contains only one of the channel endpoints, the other one shall be replaced by the null CID 0x0000.

| Reason value | Data Length | Data value |
|---|---|---|
| 0x0000 | 0 octets | N/A |
| 0x0001 | 2 octets | Actual $MTU_{sig}$ |
| 0x0002 | 4 octets | Requested CID |

*Table 4.4: Reason Data values*

## 4.2 CONNECTION REQUEST (CODE 0x02)

Connection request packets are sent to create an L2CAP channel between two devices. The L2CAP channel shall be established before configuration begins. Figure 4.4 illustrates a Connection Request packet.

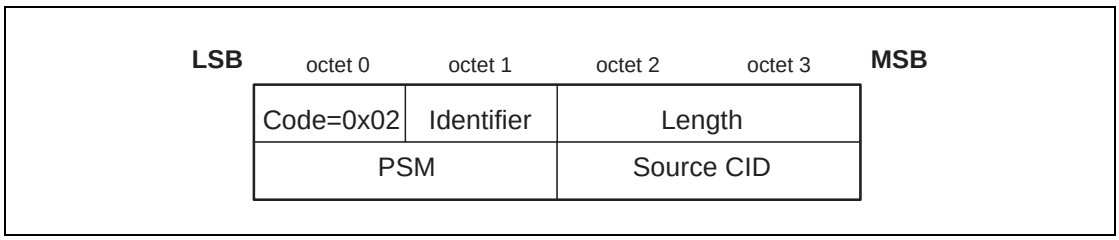



*Figure 4.4: Connection Request Packet*

The data fields are:

- *Protocol/Service Multiplexer - PSM (2 octets (minimum))*

  The PSM field is at least two octets in length. The structure of the PSM field is based on the ISO 3309 extension mechanism for address fields. All PSM values shall be ODD, that is, the least significant bit of the least significant octet must be ’1’. Also, all PSM values shall have the least significant bit of the most significant octet equal to ’0’. This allows the PSM field to be extended beyond 16 bits. PSM values are separated into two ranges. Valid values in the first range are assigned by the Bluetooth SIG and indicate protocols. The second range of values are dynamically allocated and used in conjunction with the Service Discovery Protocol (SDP). The dynamically assigned values may be used to support multiple implementations of a particular protocol.

  PSM values are defined in the Assigned Numbers document.

| Range | Type | Server Usage | Client Usage |
|---|---|---|---|
| 0x0001-0x0EFF (Note [1]) | Fixed, SIG assigned | PSM is fixed for all implementations. | PS may be obtained via SDP or may be assumed for a fixed service. Protocol used is indicated by the PSM as defined in the Assigned Numbers page. |
| >0x1000 | Dynamic | PSM may be fixed for a given implementation or may be assigned at the time the service is registered in SDP. | PSM shall be obtained via SDP upon every reconnection. PSM for one direction will typically be different from the other direction |

1. PSMs shall be odd and the least significant bit of the most significant byte shall be zero, hence the following ranges do not contain valid PSMs: 0x0100-0x01FF, 0x0300-0x03FF, 0x0500-0x05FF, 0x0700-0x07FF, 0x0900-0x09FF, 0x0B00-0x0BFF, 0x0D00-0x0DFF, 0x0F00-0x0FFF. All even values are also not valid as PSMs.

- *Source CID - SCID (2 octets)*

  The source CID is two octets in length and represents a channel endpoint on the device sending the request. Once the channel has been configured, data packets flowing to the sender of the request shall be sent to this CID. Thus, the Source CID represents the channel endpoint on the device sending the request and receiving the response.

## 4.3 CONNECTION RESPONSE (CODE 0x03)

When a device receives a Connection Request packet, it shall send a Connection Response packet. The format of the connection response packet is shown in Figure 4.5.

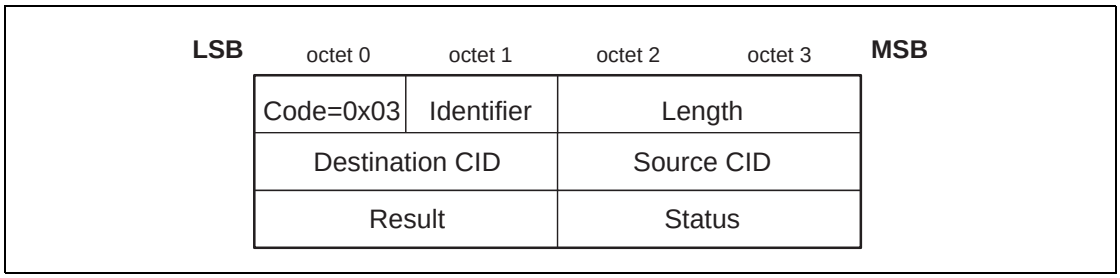



*Figure 4.5: Connection Response Packet*

The data fields are:

- *Destination Channel Identifier - DCID (2 octets)*

  This field contains the channel endpoint on the device sending this Response packet. Thus, the Destination CID represents the channel endpoint on the device receiving the request and sending the response.

- *Source Channel Identifier - SCID (2 octets)*

  This field contains the channel endpoint on the device receiving this Response packet. This is copied from the SCID field of the connection request packet.

- *Result (2 octets)*

  The result field indicates the outcome of the connection request. The result value of 0x0000 indicates success while a non-zero value indicates the connection request failed or is pending. A logical channel is established on the receipt of a successful result. Table 4.5 on page 59 defines values for this field. The DCID and SCID fields shall be ignored when the result field indicates the connection was refused.

| Value | Description |
|---|---|
| 0x0000 | Connection successful. |
| 0x0001 | Connection pending |

*Table 4.5: Result values*

| | |
|---|---|
| 0x0002 | Connection refused – PSM not supported. |
| 0x0003 | Connection refused – security block. |
| 0x0004 | Connection refused – no resources available. |
| Other | Reserved. |

*Table 4.5: Result values*

- *Status (2 octets)*

  Only defined for Result = Pending. Indicates the status of the connection. The status is set to one of the values shown in Table 4.6 on page 60.

| Value | Description |
|---|---|
| 0x0000 | No further information available |
| 0x0001 | Authentication pending |
| 0x0002 | Authorization pending |
| Other | Reserved |

*Table 4.6: Status values*

## 4.4 CONFIGURATION REQUEST (CODE 0x04)

Configuration Request packets are sent to establish an initial logical link transmission contract between two L2CAP entities and also to re-negotiate this contract whenever appropriate. The contract consists of a set of configuration parameter options defined in Section 5 on page 79. All parameter options have default values and can have previously agreed values which are values that were accepted in a previous configuration process or in a previous step in the current configuration process. The only parameters that should be included in the Configuration Request packet are those that require different values than the default or previously agreed values.

If no parameters need to be negotiated or specified then no options shall be inserted and the continuation flag (C) shall be set to zero. Any missing configuration parameters are assumed to have their most recently explicitly or implicitly accepted values. Even if all default values are acceptable, a Configuration Request packet with no options shall be sent. Implicitly accepted values are default values for the configuration parameters that have not been explicitly negotiated for the specific channel under configuration.

See Section 7.1 on page 120 for details of the configuration procedure.

Figure 4.6 defines the format of the Configuration Request packet.

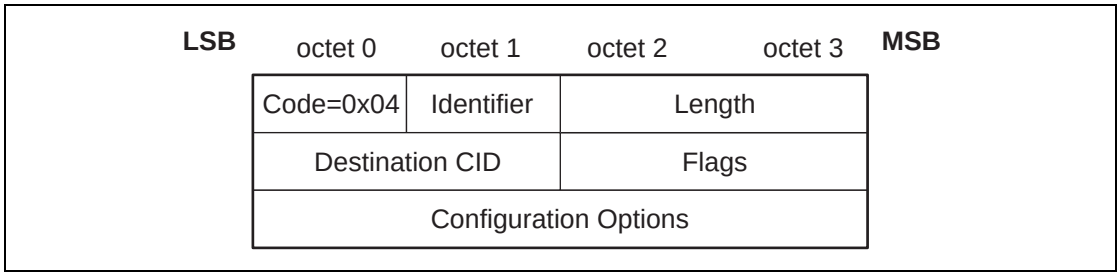



*Figure 4.6: Configuration Request Packet*

The data fields are:

- *Destination CID - DCID (2 octets)*

  This field contains the channel endpoint on the device receiving this Request packet.

- *Flags (2 octets)*

  Figure 4.7 shows the two-octet Flags field. Note the most significant bit is shown on the left.

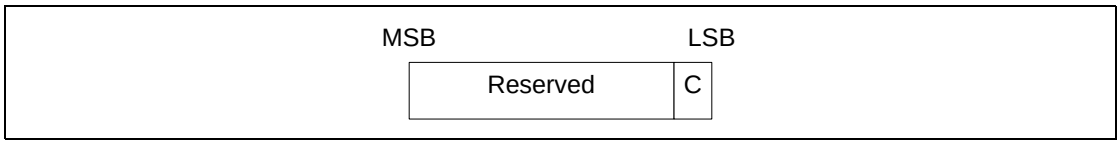



*Figure 4.7: Configuration Request Flags field format*

Only one flag is defined, the Continuation flag (C).

When both L2CAP entities support the Extended Flow Specification option, the Continuation flag shall not be used and shall be set to zero in all Configuration Request and Response packets.

When all configuration options cannot fit into a Configuration Request with length that does not exceed the receiver's $MTU_{sig}$, the options shall be passed in multiple configuration command packets. If all options fit into the receiver's $MTU_{sig}$, then they shall be sent in a single configuration request with the continuation flag set to zero. Each Configuration Request shall contain an integral number of options - partially formed options shall not be sent in a packet. Each Request shall be tagged with a different Identifier and shall be matched with a Response with the same Identifier.

When used in the Configuration Request, the continuation flag indicates the responder should expect to receive multiple request packets. The responder shall reply to each Configuration Request packet. The responder may reply to each Configuration Request with a Configuration Response containing the same option(s) present in the Request (except for those error conditions more appropriate for a Command Reject), or the responder may reply with a "Success" Configuration Response packet containing no options, delaying those options until the full Request has been received. The Configuration

Request packet with the continuation flag cleared shall be treated as the Configuration Request event in the channel state machine.

When used in the Configuration Response, the continuation flag shall be set to one if the flag is set to one in the Request. If the continuation flag is set to one in the Response when the matching Request has the flag set to zero, it indicates the responder has additional options to send to the requestor. In this situation, the requestor shall send null-option Configuration Requests (with continuation flag set to zero) to the responder until the responder replies with a Configuration Response where the continuation flag is set to zero. The Configuration Response packet with the continuation flag set to zero shall be treated as the Configuration Response event in the channel state machine.

The result of the configuration transaction is the union of all the result values. All the result values must succeed for the configuration transaction to succeed.

Other flags are reserved and shall be set to zero. L2CAP implementations shall ignore these bits.

- *Configuration Options*

  A list of the parameters and their values to be negotiated shall be provided in the Configuration Options field. These are defined in Section 5 on page 79. A Configuration Request may contain no options (referred to as an empty or null configuration request) and can be used to request a response. For an empty configuration request the length field is set to 0x0004.

## 4.5 CONFIGURATION RESPONSE (CODE 0X05)

Configuration Response packets shall be sent in reply to Configuration Request packets except when the error condition is covered by a Command Reject response. Each configuration parameter value (if any is present) in a Configuration Response reflects an 'adjustment' to a configuration parameter value that has been sent (or, in case of default values, implied) in the corresponding Configuration Request. For example, if a configuration request relates to traffic flowing from device A to device B, the sender of the configuration response may adjust this value for the same traffic flowing from device A to device B, but the response cannot adjust the value in the reverse direction.

The options sent in the Response depend on the value in the Result field. Figure 4.8 on page 63 defines the format of the Configuration Response packet. See also Section 7.1 on page 120 for details of the configuration process.

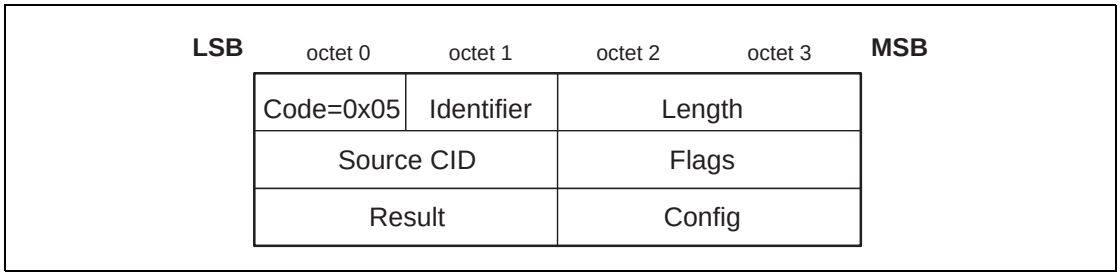



*Figure 4.8: Configuration Response Packet*

The data fields are:

- *Source CID - SCID (2 octets)*

  This field contains the channel endpoint on the device receiving this Response packet. The device receiving the Response shall check that the Identifier field matches the same field in the corresponding configuration request command and the SCID matches its local CID paired with the original DCID.

- *Flags (2 octets)*

  Figure 4.9 displays the two-octet Flags field. Note the most significant bit is shown on the left.

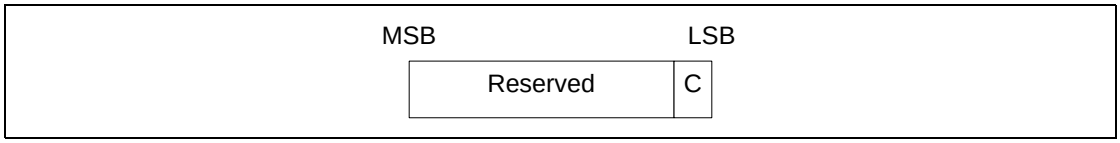



*Figure 4.9: Configuration Response Flags field format*

Only one flag is defined, the Continuation flag (C).

When both L2CAP entities support the Extended Flow Specification option, the Continuation flag shall not be used and shall be set to zero in all Configuration Request and Response packets.

More configuration responses will follow when C is set to one. This flag indicates that the parameters included in the response are a partial subset of parameters being sent by the device sending the Response packet.

The other flag bits are reserved and shall be set to zero. L2CAP implementations shall ignore these bits.

- *Result (2 octets)*

  The Result field indicates whether or not the Request was acceptable. See Table 4.7 on page 63 for possible result codes.

| Result | Description |
|---|---|
| 0x0000 | Success |

*Table 4.7: Configuration Response Result codes*

| | |
|---|---|
| 0x0001 | Failure – unacceptable parameters |
| 0x0002 | Failure – rejected (no reason provided) |
| 0x0003 | Failure – unknown options |
| 0x0004 | Pending |
| 0x0005 | Failure - flow spec rejected |
| Other | Reserved |

*Table 4.7: Configuration Response Result codes*

- *Configuration Options*

  This field contains the list of parameters being configured. These are defined in Section 5 on page 79. On a successful result (Result=0x0000) and pending result (Result=0x0004), these parameters contain the return values for any wild card parameter values (see Section 5.3 on page 82) and “adjustments” to non-negotiated configuration parameter values contained in the request. A response with the result code of 0x0000 (Success) is also referred to as a positive response.

  On an unacceptable parameters failure (Result=0x0001) the rejected parameters shall be sent in the response with the values that would have been accepted if sent in the original request. Any missing configuration parameters in the Configuration Request are assumed to have their default value or previously agreed value and they too shall be included in the Configuration Response if they need to be changed. A response with the result code of 0x0001 is also referred to as a negative response.

  On an unknown option failure (Result=0x0003), the option(s) that contain an option type field that is not understood by the recipient of the Request shall be included in the Response unless they are hints. Hints are those options in the Request that are skipped if not understood (see Section 5 on page 79). Hints shall not be included in the Response and shall not be the sole cause for rejecting the Request.

  On a flow spec rejected failure (Result=0x0005), an Extended Flow Spec option may be included to reflect the QoS level that would be acceptable (see Section 7.1.3 on page 122).

  The decision on the amount of time (or messages) spent arbitrating the channel parameters before terminating the negotiation is implementation specific.

## 4.6 DISCONNECTION REQUEST (CODE 0x06)

Terminating an L2CAP channel requires that a disconnection request be sent and acknowledged by a disconnection response. Figure 4.10 on page 65 shows a disconnection request. The receiver shall ensure that both source and destination CIDs match before initiating a channel disconnection.

Once a Disconnection Request is issued, all incoming data in transit on this L2CAP channel shall be discarded and any new additional outgoing data shall

be discarded. Once a disconnection request for a channel has been received, all data queued to be sent out on that channel shall be discarded.

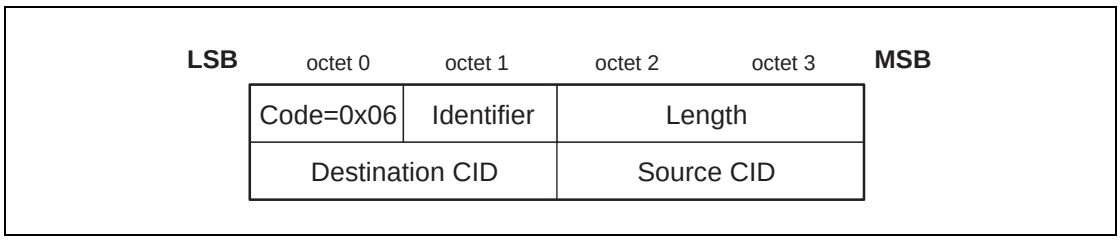



*Figure 4.10: Disconnection Request Packet*

The data fields are:

- *Destination CID - DCID (2 octets)*

  This field specifies the endpoint of the channel to be disconnected on the device receiving this request.

- *Source CID - SCID (2 octets)*

  This field specifies the endpoint of the channel to be disconnected on the device sending this request.

The SCID and DCID are relative to the sender of this request and shall match those of the channel to be disconnected. If the DCID is not recognized by the receiver of this message, a CommandReject message with 'invalid CID' result code shall be sent in response. If the receiver finds a DCID match but the SCID fails to find the same match, the request should be silently discarded.

## 4.7 DISCONNECTION RESPONSE (CODE 0x07)

Disconnection responses shall be sent in response to each valid disconnection request.

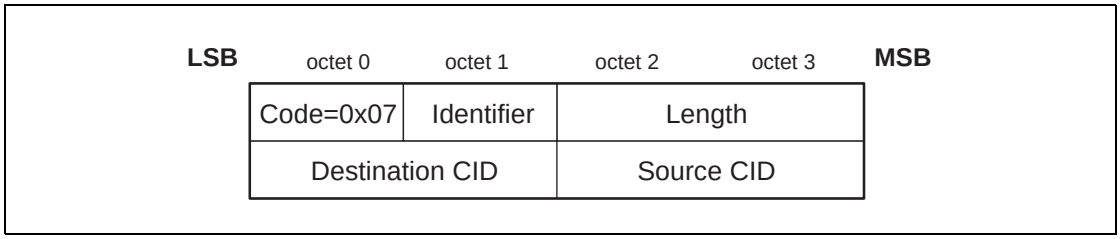



*Figure 4.11: Disconnection Response Packet*

The data fields are:

- *Destination CID - DCID (2 octets)*

  This field identifies the channel endpoint on the device sending the response.

- *Source CID - SCID (2 octets)*

This field identifies the channel endpoint on the device receiving the response.

The DCID and the SCID (which are relative to the sender of the request), and the Identifier fields shall match those of the corresponding disconnection request command. If the CIDs do not match, the response should be silently discarded at the receiver.

## 4.8 ECHO REQUEST (CODE 0x08)

Echo requests are used to request a response from a remote L2CAP entity. These requests may be used for testing the link or for passing vendor specific information using the optional data field. L2CAP entities shall respond to a valid Echo Request packet with an Echo Response packet. The Data field is optional and implementation specific. L2CAP entities should ignore the contents of this field if present.

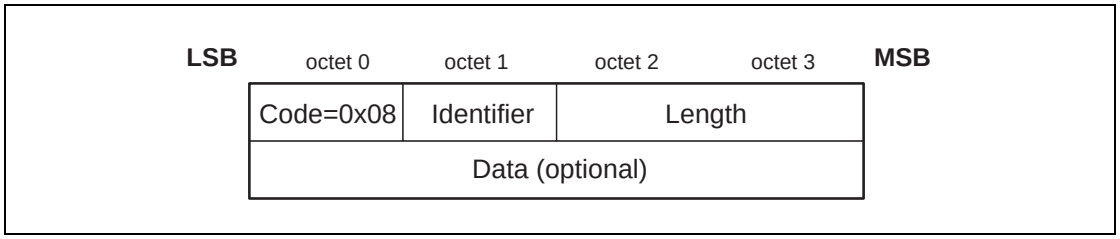



*Figure 4.12: Echo Request Packet*

## 4.9 ECHO RESPONSE (CODE 0x09)

An Echo response shall be sent upon receiving a valid Echo Request. The identifier in the response shall match the identifier sent in the Request. The optional and implementation specific data field may contain the contents of the data field in the Request, different data, or no data at all.

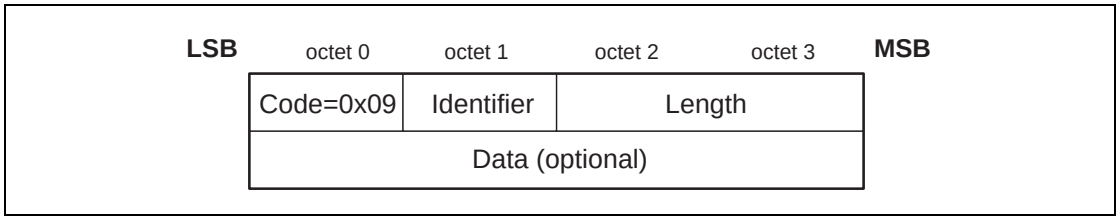



*Figure 4.13: Echo Response Packet*

## 4.10 INFORMATION REQUEST (CODE 0X0A)

Information requests are used to request implementation specific information from a remote L2CAP entity. L2CAP implementations shall respond to a valid Information Request with an Information Response. It is optional to send Information Requests.

An L2CAP implementation shall only use optional features or attribute ranges for which the remote L2CAP entity has indicated support through an Information Response. Until an Information Response which indicates support for optional features or ranges has been received only mandatory features and ranges shall be used.

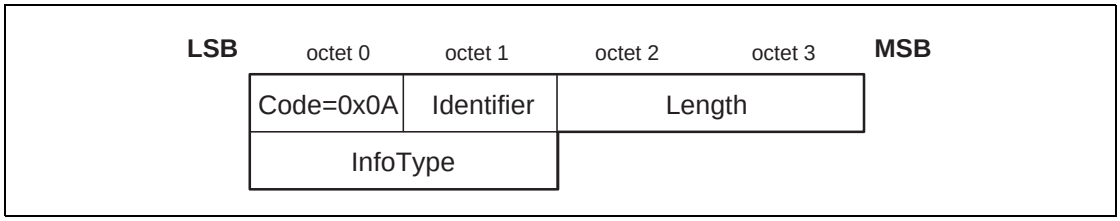



*Figure 4.14: Information Request Packet*

The data fields are:

- *InfoType (2 octets)*

  The InfoType defines the type of implementation specific information being requested. See Section 4.11 on page 67 for details on the type of information requested.

| Value | Description |
|---|---|
| 0x0001 | Connectionless MTU |
| 0x0002 | Extended features supported |
| 0x0003 | Fixed Channels supported |
| Other | Reserved |

*Table 4.8: InfoType definitions*

L2CAP entities shall not send an Information Request packet with InfoType 0x0003 over Fixed Channel CID 0x0001 until first verifying that the Fixed Channels bit is set in the Extended feature mask of the remote device. Support for fixed channels is mandatory for devices with BR/EDR/LE or LE Controllers. Information Request and Information Response shall not be used over Fixed Channel CID 0x0005.

## 4.11 INFORMATION RESPONSE (CODE 0X0B)

An information response shall be sent upon receiving a valid Information Request. The identifier in the response shall match the identifier sent in the Request. The data field shall contain the value associated with the InfoType field sent in the Request, or shall be empty if the InfoType is not supported.

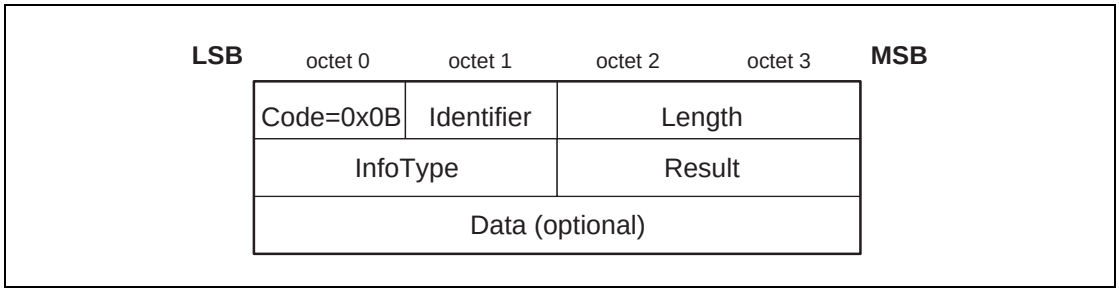



*Figure 4.15: Information Response Packet*

The data fields are:

- *InfoType (2 octets)*

  The InfoType defines the type of implementation specific information that was requested. This value shall be copied from the InfoType field in the Information Request.

- *Result (2 octets)*

  The Result contains information about the success of the request. If result is "Success," the data field contains the information as specified in Table 4.10 on page 69. If result is "Not supported," no data shall be returned.

| Value | Description |
|---|---|
| 0x0000 | Success |
| 0x0001 | Not supported |
| Other | Reserved |

*Table 4.9: Information Response Result values*

- *Data (0 or more octets)*

  The contents of the Data field depends on the InfoType.
  For InfoType = 0x0001 the data field contains the remote entity's 2-octet acceptable connectionless MTU. The default value is defined in Section 3.2 on page 45.

  For InfoType = 0x0002, the data field contains the 4 octet L2CAP extended feature mask. The feature mask refers to the extended features that the L2CAP entity sending the Information Response supports. The feature bits contained in the L2CAP feature mask are specified in Section 4.12 on page 69.

  For InfoType = 0x0003, the data field contains an 8 octet bit map that indicates which Fixed L2CAP Channels (i.e., the L2CAP channels that use a CID from 0x0000 to 0x003F) are supported. A list of available fixed channels is provided in Table 2.1 in Section 2.1. A detailed description of this InfoType data field is given in Section 4.13 on page 70.

  Note: L2CAP entities of versions prior to version 1.2, receiving an Information Request with InfoType = 0x0002 for an L2CAP feature discovery, return

an Information Response with result code "Not supported." L2CAP entities at version 1.2 to 2.1 + EDR that have an all zero extended features mask may return an Information Response with result code "Not supported."

| InfoType | Data | Data Length (octets) |
|---|---|---|
| 0x0001 | Connectionless MTU | 2 |
| 0x0002 | Extended feature mask | 4 |
| 0x0003 | Fixed Channels Supported | 8 |

*Table 4.10: Information Response Data fields*

## 4.12 EXTENDED FEATURE MASK

The features are represented as a bit mask in the Information Response data field (see Section 4.11 on page 67). For each feature a single bit is specified which shall be set to 1 if the feature is supported and set to 0 otherwise. All unknown, reserved, or unassigned feature bits shall be set to 0.

The feature mask shown in Table 4.11 on page 69 consists of 4 octets (numbered octet 0 ... 3), with bit numbers 0 ... 7 each. Within the Information Response packet data field, bit 0 of octet 0 is aligned leftmost, bit 7 of octet 3 is aligned rightmost.

Note: the L2CAP feature mask was introduced in Bluetooth v1.2 and contains features introduced after Bluetooth v1.1.

| No. | Supported feature | Octet | Bit |
|---|---|---|---|
| 0 | Flow control mode | 0 | 0 |
| 1 | Retransmission mode | 0 | 1 |
| 2 | Bi-directional QoS[1] | 0 | 2 |
| 3 | Enhanced Retransmission Mode | 0 | 3 |
| 4 | Streaming Mode | 0 | 4 |
| 5 | FCS Option | 0 | 5 |
| 6 | Extended Flow Specification for BR/EDR | 0 | 6 |
| 7 | Fixed Channels | 0 | 7 |
| 8 | Extended Window Size | 1 | 0 |
| 9 | Unicast Connectionless Data Reception | 1 | 1 |
| 31 | Reserved for feature mask extension | 3 | 7 |

*Table 4.11: Extended feature mask.*

1. Peer side supports upper layer control of the Link Manager's Bi-directional QoS, see Section 5.3 on page 82 for more details.

## 4.13 FIXED CHANNELS SUPPORTED

Each Fixed Channel is represented by a single bit in an 8 octet bit mask. The bit associated with a channel shall be set to 1 if the L2CAP entity supports signaling on that channel. The L2CAP signaling channel is mandatory and therefore the bit associated with that channel shall be set to 1.Table 4.12 shows the format of the bit mask.

| CID | Fixed Channel | Value | Octet | Bit |
|---|---|---|---|---|
| 0x0000 | Null identifier | Shall be set to 0 | 0 | 0 |
| 0x0001 | L2CAP Signaling channel | Shall be set to 1 | 0 | 1 |
| 0x0002 | Connectionless reception | 0 – if not supported<br>1 – if supported | 0 | 2 |
| 0x0003 | AMP Manager Protocol Channel | 0 – if not supported<br>1 – if supported | 0 | 3 |
| 0x0004 - 0x003E | Reserved | Shall be set to 0 and ignored upon receipt. | 0<br>1-6<br>7 | 4-7<br>0-7<br>0-6 |
| 0x003F | AMP Test Manager | 0 – if not supported<br>1 – if supported | 7 | 7 |

*Table 4.12: Fixed Channels Supported bit mask*

An L2CAP entity shall not transmit on any fixed channel (with the exception of the L2CAP signaling channel) until it has received a Fixed Channels Supported InfoType from the peer L2CAP entity indicating support for the channel, or has received a valid signaling packet from the remote device on the Fixed channel. All packets received on a non-supported fixed channel shall be ignored.

## 4.14 CREATE CHANNEL REQUEST (CODE 0X0C)

The Create Channel request packet, shown in Figure 4.16, is sent to create an L2CAP channel between two devices over the Controller identified by the Controller ID.

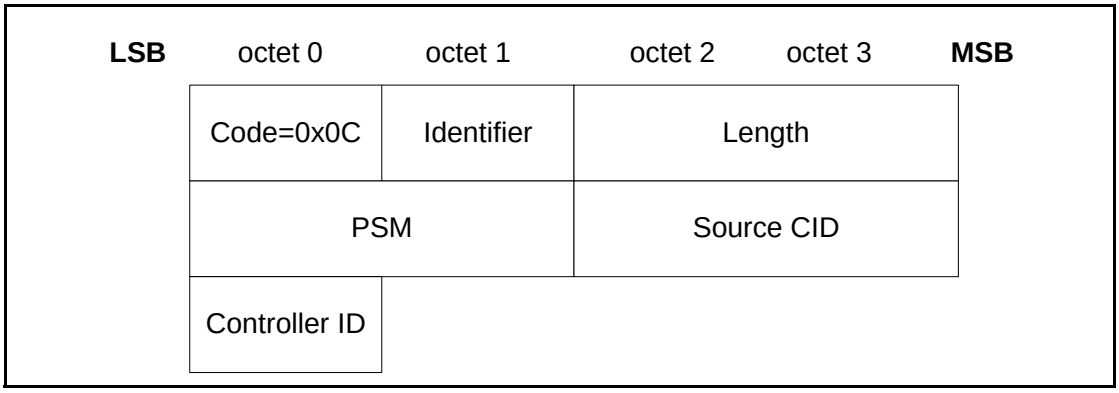



*Figure 4.16: Create Channel Request Packet*

The data fields are:

- *Protocol/Service Multiplexer –PSM (2 octets (minimum))*

  The PSM field is at least two octets in length. The structure of the PSM field is based on the ISO 3309 extension mechanism for address fields. All PSM values shall be *odd*, that is, the least significant bit of the least significant octet must be '1'. Also, all PSM values shall have the least significant bit of the most significant octet equal to '0'. This allows the PSM field to be extended beyond 16 bits. PSM values are separated into two ranges. Values in the first range are assigned by the Bluetooth SIG and indicate protocols. The second range of values are dynamically allocated and used in conjunction with the Service Discovery Protocol (SDP). The dynamically assigned values may be used to support multiple implementations of a particular protocol. PSM values are defined in the Assigned Numbers document.

- *Source CID –SCID (2 octets)*

  The source CID is two octets in length and represents a channel endpoint on the device sending the request. Once the channel has been configured, data packets flowing to the sender of the request shall be sent to this CID. Thus, the Source CID represents the channel endpoint on the device sending the request and receiving the response.

- *Controller Identifier –Controller ID (1 octet)*

  The Controller ID is one octet in length and represents the Controller physical link over which the channel shall be created. The Controller ID is the identifier of the Controller on the remote device obtained via an AMP Manager Discover Available AMPs request. See Part E, AMP Manager Protocol Specification on page 439 The Controller physical link shall already exist.

## 4.15 CREATE CHANNEL RESPONSE (CODE 0X0D)

When a device receives a Create Channel Request Packet, it shall send a Create Channel Response packet. The format of the Create Channel Response packet is shown in Figure 4.17.

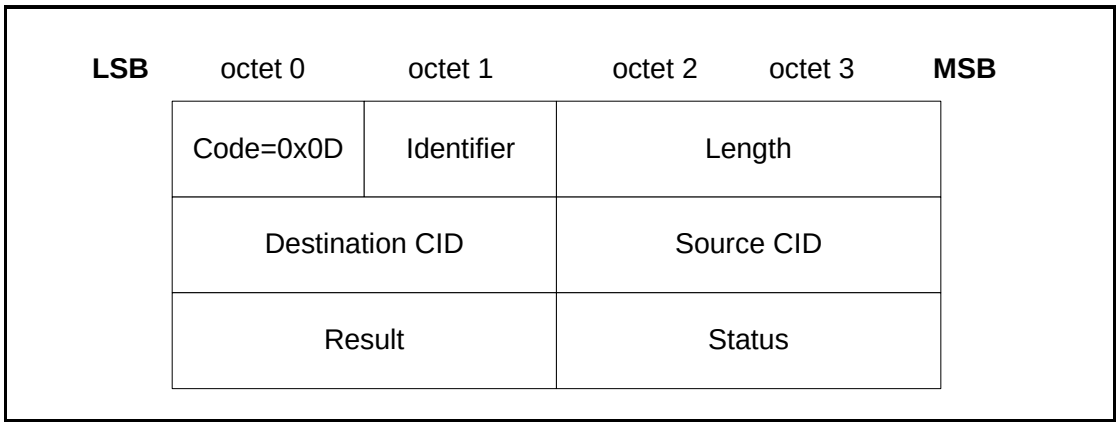



*Figure 4.17: Create Channel Response Packet*

The data fields are:

- *Destination Channel Identifier–DCID (2 octets)*

  This field contains the channel endpoint on the device sending this Response packet. Thus, the Destination CID represents the channel endpoint on the device receiving the request and sending the response.

- *Source Channel Identifier–SCID (2 octets)*

  This field contains the channel endpoint on the device receiving this Response packet. This is copied from the SCID field of the Create Channel Request packet.

- *Result (2 octets)*

  The result field indicates the outcome of the Create Channel Request. The result value of 0x0000 indicates success while a non-zero value indicates the Create Channel Request failed or is pending. A logical channel is established on the receipt of a successful result. Table 4.13 defines values for this field. The DCID and SCID fields shall be ignored when the result field indicates the connection was refused.

| Result | Description |
|---|---|
| 0x0000 | Connection successful |
| 0x0001 | Connection pending |
| 0x0002 | Connection refused - PSM not supported |
| 0x0003 | Connection refused - security block |
| 0x0004 | Connection refused - no resources available |
| 0x0005 | Connection refused - Controller ID not supported |
| Other | Reserved |

*Table 4.13: Result values*

- *Status (2 octets)*

Only defined for Result = Pending. Indicates the status of the connection. The status is set to one of the values shown in Table 4.14.

| Value | Description |
|---|---|
| 0x0000 | No further information available |
| 0x0001 | Authentication pending |
| 0x0002 | Authorization pending |
| Other | Reserved |

*Table 4.14: Status values*

## 4.16 MOVE CHANNEL REQUEST (CODE 0X0E)

Move Channel request packets are sent to move an existing L2CAP channel from a physical link on one Controller to a physical link on another Controller. The CIDs for the L2CAP channel are not changed by a Move operation. Figure 4.18 illustrates a Move Channel request packet. Channels shall only be moved if configured to use Enhanced Retransmission mode or Streaming mode. Fixed channels (see Section 2.1) shall not be moved.

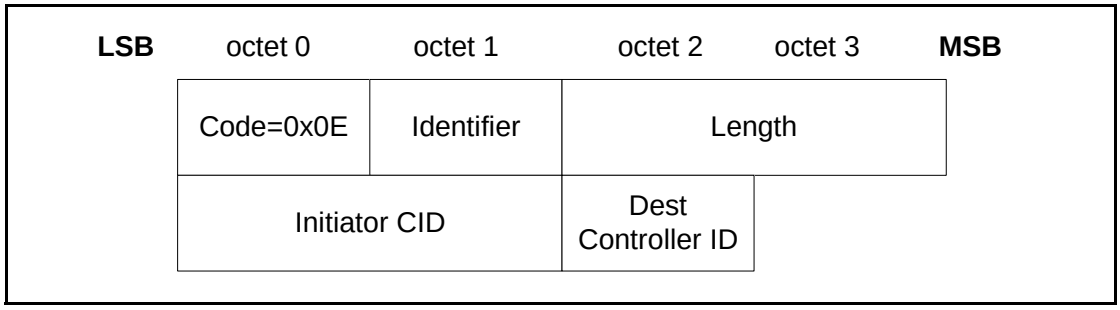



*Figure 4.18: Move Channel Request Packet*

The data fields are:

- *Initiator Channel Identifier - ICID (2 octets)*

  This field contains the channel endpoint on the device sending this Request packet.

- *Destination Controller Identifier - Dest Controller ID (1 octet)*

  The Dest Controller ID is one octet in length and represents the Controller physical link to which the channel shall be moved. The Controller ID is the identifier of the Controller on the remote device obtained via an AMP Manager Discover Available AMPs request. See Part E, AMP Manager Protocol Specification on page 439.The Controller physical link shall already exist.

## 4.17 MOVE CHANNEL RESPONSE (CODE 0X0F)

When a device receives a Move Channel request packet it shall send a Move Channel response packet. If the device has sent its own Move Channel request then a collision has occurred. One of the requests shall be rejected while the other shall proceed. The Move Channel request that shall be rejected is based on the algorithm in Section 7.7

The format of Move Channel response packet is shown in Figure 4.19.

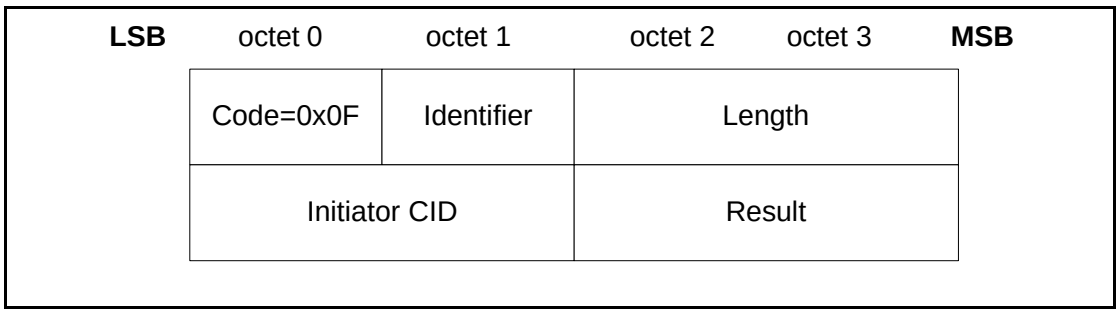



*Figure 4.19: Move Channel Response Packet*

The data fields are:

- *Initiator Channel Identifier – ICID (2 octets)*

  This field contains the channel endpoint on the device that sent the Move Channel Request (device receiving this Response packet). This is the same value as was sent in the Move Channel Request packet.

- *Result (2 octets)*

  The result field indicates the outcome of the move request. The result value of 0x0000 indicates success while a non-zero value indicates the move request failed. A logical channel is moved on the receipt of a successful result. Table 4.15 defines values for this field. The ICID field shall be ignored when the result field indicates the move was refused.

| Result | Description |
|---|---|
| 0x0000 | Move Success |
| 0x0001 | Move Pending |
| 0x0002 | Move refused - Controller ID not supported |
| 0x0003 | Move refused - new Controller ID is same as old Controller ID |
| 0x0004 | Move refused - Configuration not supported |
| 0x0005 | Move refused - Move Channel collision |
| 0x0006 | Move refused - Channel not allowed to be moved |

*Table 4.15: Result values*

| Result | Description |
|---|---|
| Other | Reserved |

*Table 4.15: Result values*

## 4.18 MOVE CHANNEL CONFIRMATION (CODE 0X10)

When the initiator of a Move Channel request receives a Move Channel response with a result code other than "pending" from the responder, it shall send a Move Channel confirmation packet, that will conclude the Move Channel procedure, informing the responder that all the QoS parameters were successfully (or not) negotiated. The format of Move Channel confirmation packet is shown in Figure 4.20.

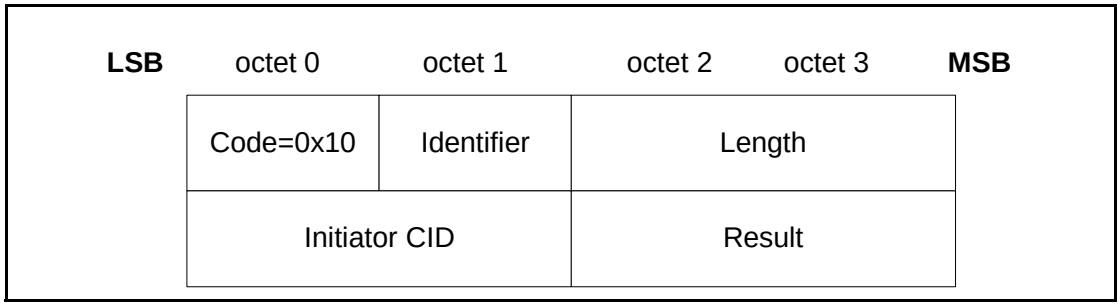



*Figure 4.20: Move Channel Confirmation Packet*

The data fields are:

- *Initiator Channel Identifier – ICID (2 octets)*

  This field contains the channel endpoint on the device sending the Move Channel Confirmation packet. This is the same value as was sent in the original Move Channel Request packet.

- *Result (2 octets)*

  The result field indicates the outcome of the move confirmation. The result value of 0x0000 indicates success while a non-zero value indicates the move failed.

| Result | Description |
|---|---|
| 0x0000 | Move success - both sides succeed |
| 0x0001 | Move failure - one or both sides refuse |
| Other | Reserved |

Table 4.16: Result values

## 4.19 MOVE CHANNEL CONFIRMATION RESPONSE (CODE 0X11)

When a device receives a Move Channel Confirmation packet it shall send a Move Channel Confirmation response packet. The purpose of the Move Channel Confirmation Response is so that Move Channel Confirmation is consistent with the command/response style. The format of Move Channel Confirmation response packet is shown in Figure 4.21.

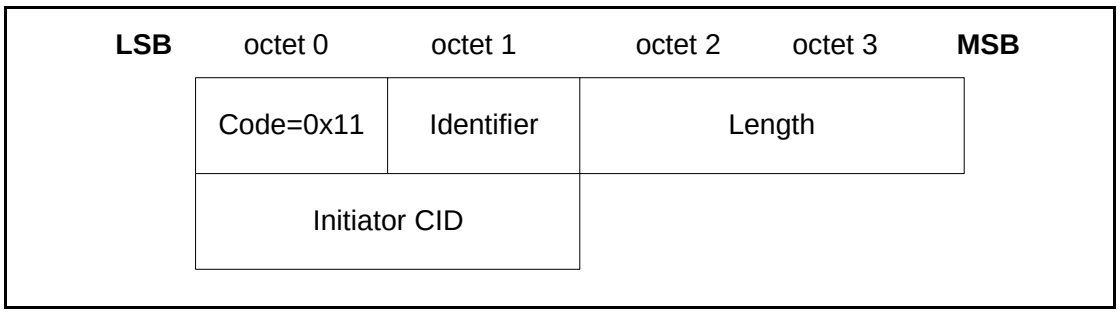



*Figure 4.21: Move Channel Confirmation Response Packet*

The data field is:

- *Initiator Channel Identifier – ICID (2 octets)*

  This field contains the channel endpoint on the device that sent the Move Channel Confirmation packet.

## 4.20 CONNECTION PARAMETER UPDATE REQUEST (CODE 0X12)

This command shall only be sent from the LE slave device to the LE master device. If an LE slave Host receives a Connection Parameter Update Request packet it shall respond with a Command Reject packet with reason 0x0000 (Command not understood).

The Connection Parameter Update Request allows the LE slave Host to request a set of new connection parameters. When the LE master Host receives a Connection Parameter Update Request packet, depending on the parameters of other connections, the LE master Host may accept the requested parameters and deliver the requested parameters to its Controller or reject the request. In devices supporting HCI the LE master Host delivers the requested parameters to its Controller using the HCI_LE_Connection_Update command (see [Vol. 2] Part E, Section 7.8.18). If the LE master Host accepts the requested parameters it shall send the Connection Parameter Update Response packet with result 0x0000 (Parameters accepted) otherwise it shall set the result to 0x0001 (request rejected).

The LE slave Host will receive an indication from the LE slave Controller when the connection parameters have been updated. In devices supporting HCI this

notification will be in the form of an LE Connection Update Complete event (see [Vol. 2] Part E, Section 7.7.65.1). If the LE master Controller rejects the updated connection parameters no indication from the LE slave Controller will be sent to the LE slave Host.

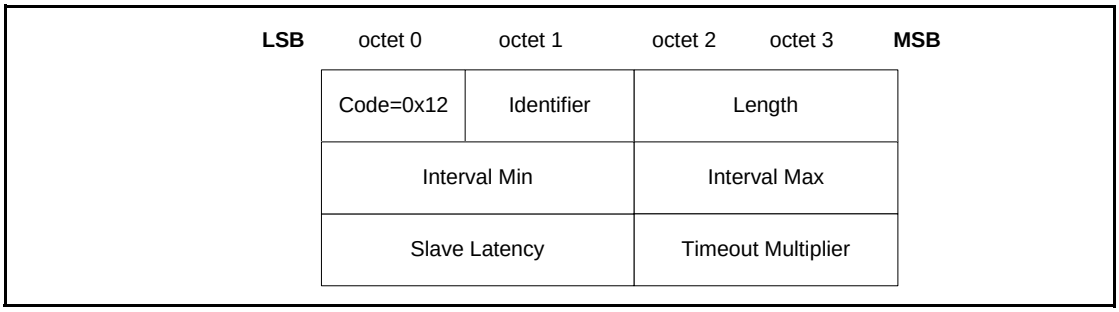



*Figure 4.22: Connection Parameters Update Request packet*

The data fields are:

- *Interval Min (2 octets)*

  Defines minimum value for the connection event interval in the following manner:
  *connIntervalMin* = Interval Min * 1.25 ms. Interval Min range: 6 to 3200 frames where 1 frame is 1.25 ms and equivalent to 2 BR/EDR slots. Values outside the range are reserved. Interval Min shall be less than or equal to Interval Max.

- *Interval Max (2 octets)*

  Defines maximum value for the connection event interval in the following manner:
  *connIntervalMax* = Interval Max * 1.25 ms. Interval Max range: 6 to 3200 frames. Values outside the range are reserved. Interval Max shall be equal to or greater than the Interval Min.

- *Slave Latency (2 octets)*

  Defines the slave latency parameter (as number of LL connection events) in the following manner:

  *connSlaveLatency* = Slave Latency. The Slave Latency field shall have a value in the range of 0 to ((*connSupervisionTimeout* / *connIntervalMax*) - 1). The Slave Latency field shall be less than 500.

- *Timeout Multiplier (2 octets)*

  Defines connection timeout parameter in the following manner:

  *connSupervisionTimeout* = Timeout Multiplier * 10 ms

  The Timeout Multiplier field shall have a value in the range of 10 to 3200.

## 4.21 CONNECTION PARAMETER UPDATE RESPONSE (CODE 0X13)

This response shall only be sent from the LE master device to the LE slave device.

The Connection Parameter Update Response packet shall be sent by the master Host when it receives a Connection Parameter Update Request packet. If the LE master Host accepts the request it shall send the connection parameter update to its Controller.

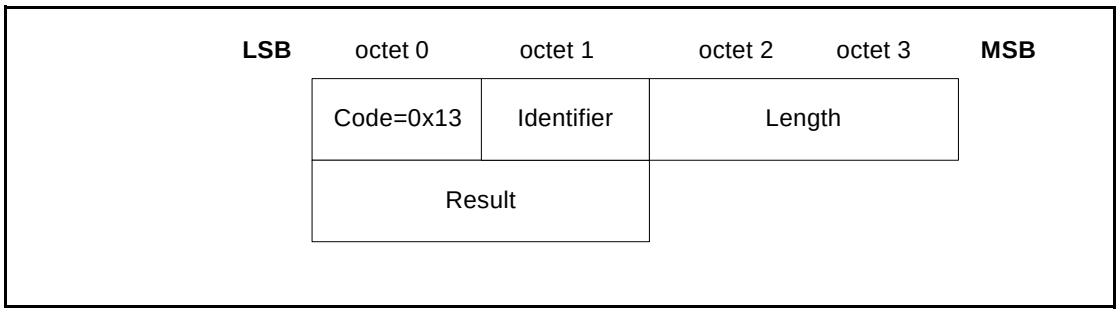



*Figure 4.23: Connection Parameters Update Response packet*

The data fields are:

- *Result (2 octets)*

  The result field indicates the response to the Connection Parameter Update Request. The result value of 0x0000 indicates that the LE master Host has accepted the connection parameters while 0x0001 indicates that the LE master Host has rejected the connection parameters

.

| Value | Description |
|---|---|
| 0x0000 | Connection Parameters accepted |
| 0x0001 | Connection Parameters rejected |
| Other | Reserved |

# 5 CONFIGURATION PARAMETER OPTIONS

Options are a mechanism to extend the configuration parameters. Options shall be transmitted as information elements containing an option type, an option length, and one or more option data fields. Figure 5.1 illustrates the format of an option.

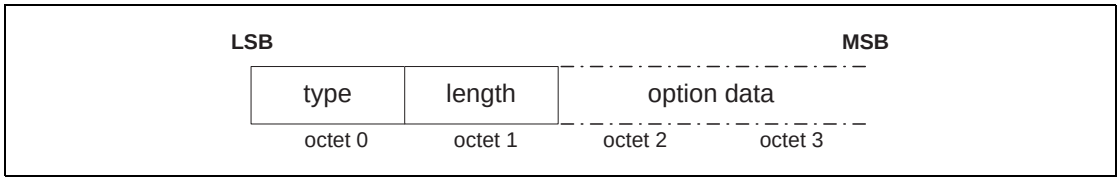



*Figure 5.1: Configuration option format*

The configuration option fields are:

- *Type (1 octet)*

  The option type field defines the parameters being configured. The most significant bit of the type determines the action taken if the option is not recognized.

  0 - option must be recognized; if the option is not recognized then refuse the configuration request

  1 - option is a hint; if the option is not recognized then skip the option and continue processing

- *Length (1 octet)*

  The length field defines the number of octets in the option data. Thus an option type without option data has a length of 0.

- *Option data*

  The contents of this field are dependent on the option type.

## 5.1 MAXIMUM TRANSMISSION UNIT (MTU)

This option specifies the maximum SDU size the sender of this option is capable of accepting for a channel. The type is 0x01, and the payload length is 2 octets, carrying the two-octet MTU size value as the only information element (see Figure 5.2 on page 81). Unlike the B-Frame length field, the I-frame length field may be greater than the configured MTU because it includes the octet lengths of the Control, L2CAP SDU Length (when present), and frame check sequence fields as well as the Information octets.

MTU is not a negotiated value, it is an informational parameter that each device can specify independently. It indicates to the remote device that the local device can receive, in this channel, an MTU larger than the minimum required. All L2CAP implementations shall support a minimum MTU of 48 octets over the ACL-U logical link and 23 octets over the LE-U logical link; how-

ever, some protocols and profiles explicitly require support for a larger MTU. The minimum MTU for a channel is the larger of the L2CAP minimum 48 octet MTU and any MTU explicitly required by the protocols and profiles using that channel. (Note: the MTU is only affected by the profile directly using the channel. For example, if a service discovery transaction is initiated by a non service discovery profile, that profile does not affect the MTU of the L2CAP channel used for service discovery).

The following rules shall be used when responding to a configuration request specifying the MTU for a channel:

- A request specifying any MTU greater than or equal to the minimum MTU for the channel shall be accepted.
- A request specifying an MTU smaller than the minimum MTU for the channel may be rejected.

The signaling described in Section 4.5 on page 62 may be used to reject an MTU smaller than the minimum MTU for a channel. The "failure-unacceptable parameters" result sent to reject the MTU shall include the proposed value of MTU that the remote device intends to transmit. It is implementation specific whether the local device continues the configuration process or disconnects the channel.

If the remote device sends a positive configuration response it should include the actual MTU to be used on this channel for traffic flowing into the local device. Following the above rules, the actual MTU cannot be less than 48 bytes.This is the minimum of the MTU in the configuration request and the outgoing MTU capability of the device sending the configuration response. The new agreed value (the default value in a future re-configuration) is the value specified in the response.

Note: For backwards compatibility reception of the MTU option in a negative Configuration Response where the MTU option is not in error should be interpreted in the same way as it is in a positive Configuration Response (e.g. the case where another configuration option value is unacceptable but the negative Configuration Response contains the MTU option in addition to the unacceptable option).

The MTU to be used on this channel for the traffic flowing in the opposite direction will be established when the remote device sends its own Configuration Request as explained in Section 4.4 on page 60.

If the configured mode is Enhanced Retransmission mode or Streaming mode then MTU shall not be reconfigured to a smaller size.

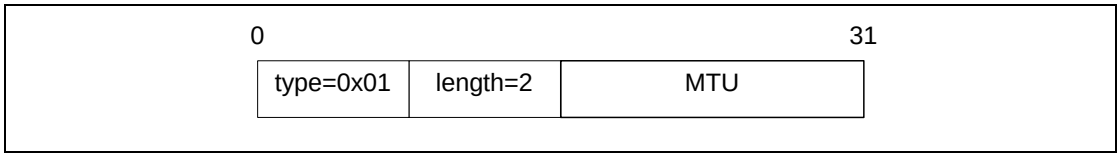



*Figure 5.2: MTU Option Format*

The option data field is:

- Maximum Transmission Unit - MTU (2 octets)

  The MTU field is the maximum SDU size, in octets, that the originator of the Request can accept for this channel. The MTU is asymmetric and the sender of the Request shall specify the MTU it can receive on this channel if it differs from the default value. L2CAP implementations shall support a minimum MTU size of 48 octets. The default value is 672 octets[2].

## 5.2 FLUSH TIMEOUT OPTION

This option is used to inform the recipient of the Flush Timeout the sender is going to use. This option shall not be used if the Extended Flow Specification is used. The Flush Timeout is defined in the BR/EDR Baseband specification “Flushing Payloads” on page 147[vol. 2]. The type is 0x02 and the payload size is 2 octets. The Flush Timeout option is negotiable.

If the remote device returns a negative response to this option and the local device cannot honor the proposed value, then it shall either continue the configuration process by sending a new request with the original value, or disconnect the channel. The flush timeout applies to all channels on the same ACL logical transport but may be overridden on a packet by packet basis by marking individual L2CAP packets as non-automatically-flushable via the Packet_Boundary_Flag in the HCI ACL Data Packet (see Section 1.1 on page 30).

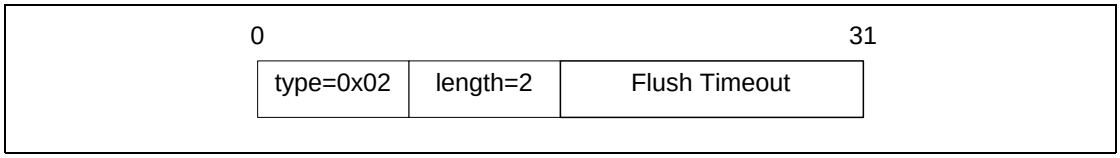



*Figure 5.3: Flush Timeout option format.*

The option data field is:

- *Flush Timeout*

2. The default MTU was selected based on the payload carried by two baseband DH5 packets (2*341=682 octets) minus the baseband ACL headers (2*2=4 octets) and a 6-octet L2CAP header. Note that the L2CAP header length is 4 octets (see Section 3.3.1) but for historical reasons an L2CAP header length of 6 bytes is used.

This value is the Flush Timeout in milliseconds. This is an asymmetric value and the sender of the Request shall specify its flush timeout value if it differs from the default value of 0xFFFF.

Possible values are:

0x0001 - no retransmissions at the baseband level should be performed since the minimum polling interval is 1.25 ms.

0x0002 to 0xFFFE - Flush Timeout used by the baseband.

0xFFFF - an infinite amount of retransmissions. This is also referred to as a 'reliable channel'. In this case, the baseband shall continue retransmissions until physical link loss is declared by link manager timeouts.

## 5.3 QUALITY OF SERVICE (QOS) OPTION

This option specifies a flow specification similar to RFC 1363[3]. Although the RFC flow specification addresses only the transmit characteristics, the Bluetooth QoS interface can handle the two directions (Tx and Rx) in the negotiation as described below.

If no QoS configuration parameter is negotiated the link shall assume the default parameters. The QoS option is type 0x03. This option shall not be used if the Extended Flow Specification option is used. The QoS option is negotiable.

In a configuration request, this option describes the outgoing traffic flow from the device sending the request. In a positive Configuration Response, this option describes the incoming traffic flow agreement to the device sending the response. In a negative Configuration Response, this option describes the preferred incoming traffic flow to the device sending the response.

L2CAP implementations are only required to support 'Best Effort' service, support for any other service type is optional. Best Effort does not require any guarantees. If no QoS option is placed in the request, Best Effort shall be assumed. If any QoS guarantees are required then a QoS configuration request shall be sent.

The remote device's Configuration Response contains information that depends on the value of the result field (see Section 4.5 on page 62). If the request was for Guaranteed Service, the response shall include specific values for any wild card parameters (see Token Rate and Token Bucket Size descriptions) contained in the request. If the result is "Failure – unacceptable parameters", the response shall include a list of outgoing flow specification parameters and parameter values that would make a new Connection Request from the local device acceptable by the remote device. Both explicitly referenced in a Configuration Request or implied configuration parameters can be included in a Configuration Response. Recall that any missing configuration parameters

3. Internet Engineering Task Force, "A Proposed Flow Specification", RFC 1363, September 1992.

from a Configuration Request are assumed to have their most recently accepted values.

If a configuration request contains any QoS option parameters set to “do not care” then the configuration response shall set the same parameters to “do not care”. This rule applies for both Best Effort and Guaranteed Service.

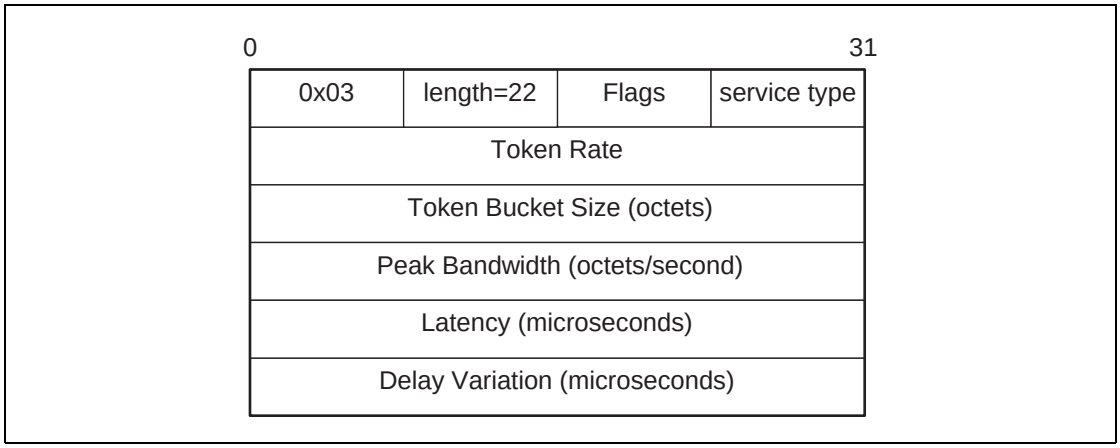



*Figure 5.4: Quality of Service (QoS) option format containing Flow Specification*

The option data fields are:

- *Flags (1 octet)*

  Reserved for future use and shall be set to 0 and ignored by the receiver.

- *Service Type (1 octet)*

  This field indicates the level of service required. Table 5.1 on page 83 defines the different services available. The default value is ‘Best effort’.

  If ‘Best effort’ is selected, the remaining parameters should be treated as optional by the remote device. The remote device may choose to ignore the fields, try to satisfy the parameters but provide no response (QoS option omitted in the Response message), or respond with the settings it will try to meet.

  If ‘No traffic’ is selected, the remainder of the fields shall be ignored because there is no data being sent across the channel in the outgoing direction.

| Value | Description |
|---|---|
| 0x00 | No traffic |
| 0x01 | Best effort (Default) |
| 0x02 | Guaranteed |
| Other | Reserved |

*Table 5.1: Service type definitions*

- *Token Rate (4 octets)*

The value of this field represents the average data rate with which the application transmits data. The application may send data at this rate continuously. On a short time scale the application may send data in excess of the average data rate, dependent on the specified Token Bucket Size and Peak Bandwidth (see below). The Token Bucket Size and Peak Bandwidth allow the application to transmit data in a 'bursty' fashion.

The Token Rate signalled between two L2CAP peers is the data transmitted by the application and shall exclude the L2CAP protocol overhead. The Token Rate signalled over the interface between L2CAP and the Link Manager shall include the L2CAP protocol overhead. Furthermore the Token Rate value signalled over this interface may also include the aggregation of multiple L2CAP channels onto the same ACL logical transport.

The Token Rate is the rate with which traffic credits are provided. Credits can be accumulated up to the Token Bucket Size. Traffic credits are consumed when data is transmitted by the application. When traffic is transmitted, and there are insufficient credits available, the traffic is non-conformant. The Quality of Service guarantees are only provided for conformant traffic. For non-conformant traffic there may not be sufficient resources such as bandwidth and buffer space. Furthermore non-conformant traffic may violate the QoS guarantees of other traffic flows.

The Token Rate is specified in octets per second. The value 0x00000000 indicates no token rate is specified. This is the default value and means “do not care”. When the Guaranteed service is selected, the default value shall not be used. The value 0xFFFFFFFF is a wild card matching the maximum token rate available. The meaning of this value depends on the service type. For best effort, the value is a hint that the application wants as much bandwidth as possible. For Guaranteed service the value represents the maximum bandwidth available at the time of the request.

- *Token Bucket Size (4 octets)*

  The Token Bucket Size specifies a limit on the 'burstiness' with which the application may transmit data. The application may offer a burst of data equal to the Token Bucket Size instantaneously, limited by the Peak Bandwidth (see below). The Token Bucket Size is specified in octets.

  The Token Bucket Size signalled between two L2CAP peers is the data transmitted by the application and shall exclude the L2CAP protocol overhead. The Token Bucket Size signalled over the interface between L2CAP and Link Manager shall include the L2CAP protocol overhead. Furthermore the Token Bucket Size value over this interface may include the aggregation of multiple L2CAP channels onto the same ACL logical transport.

  The value of 0x00000000 means that no token bucket is needed; this is the default value. When the Guaranteed service is selected, the default value shall not be used. The value 0xFFFFFFFF is a wild card matching the maximum token bucket available. The meaning of this value depends on the service type. For best effort, the value indicates the application wants a bucket as big as possible. For Guaranteed service the value represents the maximum L2CAP SDU size.

The Token Bucket Size is a property of the traffic carried over the L2CAP channel. The Maximum Transmission Unit (MTU) is a property of an L2CAP implementation. For the Guaranteed service the Token Bucket Size shall be smaller or equal to the MTU.

- *Peak Bandwidth (4 octets)*

  The value of this field, expressed in octets per second, limits how fast packets from applications may be sent back-to-back. Some systems can take advantage of this information, resulting in more efficient resource allocation.

  The Peak Bandwidth signalled between two L2CAP peers specifies the data transmitted by the application and shall exclude the L2CAP protocol overhead. The Peak Bandwidth signalled over the interface between L2CAP and Link Manager shall include the L2CAP protocol overhead. Furthermore the Peak Bandwidth value over this interface may include the aggregation of multiple L2CAP channels onto the same ACL logical transport.

  The value of 0x00000000 means "don't care." This states that the device has no preference on incoming maximum bandwidth, and is the default value. When the Guaranteed service is selected, the default value shall not be used.

- *Access Latency (4 octets)*

  The value of this field is the maximum acceptable delay of an L2CAP packet to the air-interface. The precise interpretation of this number depends on over which interface this flow parameter is signalled. When signalled between two L2CAP peers, the Access Latency is the maximum acceptable delay between the instant when the L2CAP SDU is received from the upper layer and the start of the L2CAP SDU transmission over the air. When signalled over the interface between L2CAP and the Link Manager, it is the maximum delay between the instant the first fragment of an L2CAP PDU is stored in the Host Controller buffer and the initial transmission of the L2CAP packet on the air.

  Thus the Access Latency value may be different when signalled between L2CAP and the Link Manager to account for any queuing delay at the L2CAP transmit side. Furthermore the Access Latency value may include the aggregation of multiple L2CAP channels onto the same ACL logical transport.

  The Access Latency is expressed in microseconds. The value 0xFFFFFFFF means “do not care” and is the default value. When the Guaranteed service is selected, the default value shall not be used.

- *Delay Variation (4 octets)*

  The value of this field is the difference, in microseconds, between the maximum and minimum possible delay of an L2CAP SDU between two L2CAP peers. The Delay Variation is a purely informational parameter. The value 0xFFFFFFFF means “do not care” and is the default value.

## 5.4 RETRANSMISSION AND FLOW CONTROL OPTION

This option specifies whether retransmission and flow control is used. If the feature is used both incoming and outgoing parameters are specified by this option. The Retransmission and Flow Control option contains both negotiable parameters and non-negotiable parameters. The mode parameter controls both incoming and outgoing data flow (i.e. both directions have to agree) and is negotiable. The other parameters control incoming data flow and are non-negotiable.

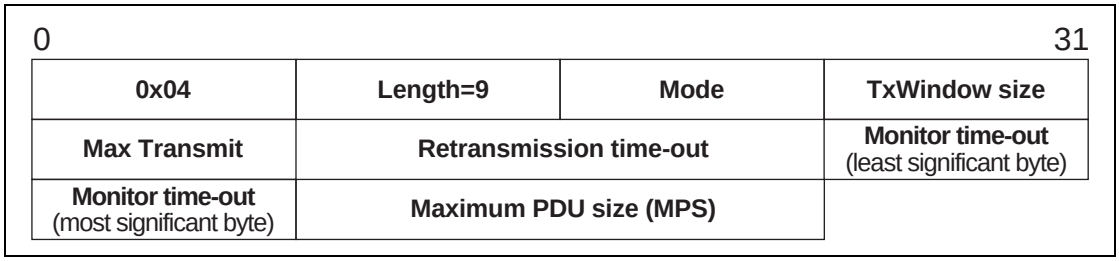



*Figure 5.5: Retransmission and Flow Control option format.*

The option data fields are:

- *Mode (1 octet)*

  The field contains the requested mode of the link. Possible values are shown in Table 5.2 on page 86.

| Value | Description |
|---|---|
| 0x00 | L2CAP Basic Mode |
| 0x01 | Retransmission mode |
| 0x02 | Flow control mode |
| 0x03 | Enhanced Retransmission mode |
| 0x04 | Streaming mode |
| Other values | Reserved for future use |

*Table 5.2: Mode definitions.*

The Basic L2CAP mode is the default. If Basic L2CAP mode is requested then all other parameters shall be ignored.

Enhanced Retransmission mode should be enabled if a reliable channel has been requested. Enhanced Retransmission mode shall only be sent to an L2CAP entity that has previously advertised support for the mode in its Extended Feature Mask (see section 4.12).

Streaming mode should be enabled if a finite L2CAP Flush Time-Out is set on an L2CAP connection. Streaming mode shall only be sent to a device that has previously advertised support for the mode in the Extended Feature Mask (see section 4.12).

Flow Control mode and Retransmission mode shall only be used for backwards compatibility with L2CAP entities that do not support Enhanced Retransmission mode or Streaming mode.

- *TxWindow size (1 octet)*

  This field specifies the size of the transmission window for Flow Control mode, Retransmission mode, and Enhanced Retransmission mode. The range is 1 to 32 for Flow Control mode and Retransmission mode. The range is 1 to 63 for Enhanced Retransmission mode.

  In Retransmission mode and Flow Control mode this parameter should be negotiated to reflect the buffer sizes allocated for the connection on both sides. In general, the Tx Window size should be made as large as possible to maximize channel utilization. Tx Window size also controls the delay on flow control action. The transmitting device can send as many PDUs fit within the window.

  In Enhanced Retransmission mode this value indicates the maximum number of I-frames that the sender of the option can receive without acknowledging some of the received frames. It is not negotiated. It is an informational parameter that each L2CAP entity can specify separately. In general, the TxWindow size should be made as large as possible to maximize channel utilization. The transmitting L2CAP entity can send as many PDUs as will fit within the receiving L2CAP entity's TxWindow. TxWindow size values in a Configuration Response indicate the maximum number of packets the sender can send before it requires an acknowledgement. In other words it represents the number of unacknowledged packets the send can hold. The value sent in a Configuration Response shall be less than or equal to the TxWindow size sent in the Configuration Request. The receiver of this option in the Configuration Response may use this value as part of its acknowledgement algorithm.

  In Streaming mode this value is not used and shall be set to 0 and ignored by the receiving device.

- *MaxTransmit (1 octet)*

  This field controls the number of transmissions of a single I-frame that L2CAP is allowed to try in Retransmission mode and Enhanced Retransmission mode. The minimum value is 1 (one transmission is permitted).

  MaxTransmit controls the number of retransmissions that L2CAP is allowed to try in Retransmission mode and Enhanced Retransmission mode before accepting that a packet and the channel is lost. When a packet is lost after being transmitted MaxTransmit times the channel shall be disconnected by sending a Disconnect request (see Section 4.6). In Enhanced Retransmission mode MaxTransmit controls the number of retransmissions for I-frames and S-frames with P-bit set to 1. The sender of the option in a Configuration Request specifies the value that shall be used by the receiver of the option. MaxTransmit values in a Configuration Response shall be ignored. Lower values might be appropriate for services requiring low latency. Higher values will be suitable for a link requiring robust operation. A value of 1 means that no retransmissions will be made but also means that the channel will be disconnected as soon as a packet is

lost. MaxTransmit shall not be set to zero in Retransmission mode. In Enhanced Retransmission mode a value of zero for MaxTransmit means infinite retransmissions.

In Streaming mode this value is not used and shall be set to 0 and ignored by the receiving L2CAP entity.

- *Retransmission time-out (2 octets)*

  This is the value in milliseconds of the retransmission time-out (this value is used to initialize the RetransmissionTimer).

  The purpose of this timer in retransmission mode is to activate a retransmission in some exceptional cases. In such cases, any delay requirements on the channel may be broken, so the value of the timer should be set high enough to avoid unnecessary retransmissions due to delayed acknowledgements. Suitable values could be 100's of milliseconds and up.

  The purpose of this timer in flow control mode is to supervise I-frame transmissions. If an acknowledgement for an I-frame is not received within the time specified by the RetransmissionTimer value, either because the I-frame has been lost or the acknowledgement has been lost, the timeout will cause the transmitting side to continue transmissions. Suitable values are implementation dependent.

  The purpose of this timer in Enhanced Retransmission mode is to detect lost I-frames and initiate appropriate error recovery. The value used for the Retransmission time-out is specified in Section 8.6.2. The value sent in a Configuration Request is also specified in Section 8.6.2. A value for the Retransmission time-out shall be sent in a positive Configuration Response and indicates the value that will be used by the sender of the Configuration Response.

  In Streaming mode this value is not used and shall be set to 0 and ignored by the receiving L2CAP entity.

- *Monitor time-out (2 octets)*

  In Retransmission mode this is the value in milliseconds of the interval at which S-frames should be transmitted on the return channel when no frames are received on the forward channel. (this value is used to initialize the MonitorTimer, see below).

  This timer ensures that lost acknowledgements are retransmitted. Its main use is to recover Retransmission Disable Bit changes in lost frames when no data is being sent. The timer shall be started immediately upon transitioning to the open state. It shall remain active as long as the connection is in the open state and the retransmission timer is not active. Upon expiration of the Monitor timer an S-frame shall be sent and the timer shall be restarted. If the monitor timer is already active when an S-frame is sent, the timer shall be restarted. An idle connection will have periodic monitor traffic sent in both directions. The value for this time-out should also be set to 100's of milliseconds or higher.

  In Enhanced Retransmission mode the Monitor time-out is used to detect lost S-frames with P-bit set to 1. If the time-out occurs before a response

with the F-bit set to 1 is received the S-frame is resent. The value used for the Monitor time-out is specified in Section 8.6.3. The value sent in a Configuration Request is also specified in Section 8.6.2. A value for the Monitor time-out shall be sent in a positive Configuration Response and indicates the value that will be used by the sender of the Configuration Response.

In Streaming mode this value is not used and shall be set to 0 and ignored by the receiving device.

- *Maximum PDU payload Size - MPS (2 octets)*

  The maximum size of payload data in octets that the L2CAP layer entity sending the option in a Configuration Request is capable of accepting, i.e. the MPS corresponds to the maximum PDU payload size. Values used for MPS should take into account all possible Controllers to which the channel might be moved given that the MPS value cannot be renegotiated once the channel is created. Each device specifies the value separately. An MPS value sent in a positive Configuration Response is the actual MPS the receiver of the Configuration Request will use on this channel for traffic flowing into the local device. An MPS value sent in a positive Configuration Response shall be equal to or smaller than the value sent in the Configuration Request.

When using Retransmission mode and Flow Control mode the settings are configured separately for the two directions of an L2CAP connection. For example, in operating with an L2CAP entity implementing an earlier version of the core specification, an L2CAP connection can be configured as Flow Control mode in one direction and Retransmission mode in the other direction. If Basic L2CAP mode is configured in one direction and Retransmission mode or Flow control mode is configured in the other direction on the same L2CAP channel then the channel shall not be used.

Note: This asymmetric configuration only occurs during configuration.

When using Enhanced Retransmission mode or Streaming mode, both directions of the L2CAP connection shall be configured to the same mode. A precedence algorithm shall be used by both devices so a mode conflict can be resolved in a quick and deterministic manner.

There are two operating states:

- A device has a desired mode but is willing to use another mode (known as "state 1"), and
- A device requires a specific mode (known as "state 2"). This includes cases where channels are created over AMP-U logical links or ACL-U logical links operating as described in Section 7.10.

In state 1, Basic L2CAP mode has the highest precedence and shall take precedence over Enhanced Retransmission mode and Streaming mode. Enhanced Retransmission mode has the second highest precedence and shall take precedence over all other modes except Basic L2CAP mode. Streaming

mode shall have the next level of precedence after Enhanced Retransmission mode.

In state 2, a layer above L2CAP requires Enhanced Retransmission mode or Streaming mode. In this case, the required mode takes precedence over all other modes.

A device does not know in which state the remote device is operating so the state 1 precedence algorithm assumes that the remote device may be a state 2 device. If the mode proposed by the remote device has a higher precedence (according to the state 1 precedence) then the algorithm will operate such that creation of a channel using the remote device's mode has higher priority than disconnecting the channel.

The algorithm for state 1 devices is divided into two parts. Part one covers the case where the remote device proposes a mode with a higher precedence than the state 1 local device. Part two covers the case where the remote device proposes a mode with a lower precedence than the state 1 local device. Part one of the algorithm is as follows:

- When the remote device receives the Configuration Request from the local device it will either reject the local device's Configuration Request by sending a negative Configuration Response or disconnect the channel. The negative Configuration Response will contain the remote device's desired mode.
- Upon receipt of the negative Configuration Response the local device shall either send a second Configuration Request proposing the mode contained in the remote device's negative Configuration Response or disconnect the channel.
- When the local device receives the Configuration Request from the remote device it shall send a positive Configuration Response or disconnect the channel.
- If the mode in the remote Device's negative Configuration Response does not match the mode in the remote device's Configuration Request then the local device shall disconnect the channel.

Part two of the algorithm is as follows:

- When the local device receives the Configuration Request from the remote device it shall reject the Configuration Request by sending a negative Configuration Response proposing its desired mode. The local device's desired mode shall be the same mode it sent in its Configuration Request. Upon receiving the negative Configuration Response the remote device will either send a second Configuration Request or disconnect the channel.
- If the local device receives a second Configuration Request from the remote device that does not contain the desired mode then the local device shall disconnect the channel.
- If the local device receives a negative Configuration Response then it shall disconnect the channel.

An example of the algorithm for state 1 devices is as follows:

- The remote device proposes Basic L2CAP mode in a Configuration Request and the local device proposes Enhanced Retransmission mode or Streaming mode. The remote device rejects the local device's Configuration Request by sending a negative Configuration Response proposing Basic mode. The local device will send a second Configuration Request proposing Basic L2CAP mode or disconnect the channel. If the local device sends a second Configuration Request that does not propose Basic L2CAP mode then the remote device will disconnect the channel. If the local device rejects the remote device's Configuration Request then the remote device will disconnect the channel.

The algorithm for state 2 devices is as follows:

- If the local device proposes a mode in a Configuration Request and the remote device proposes a different mode or rejects the local device's Configuration Request then the local device shall disconnect the channel.

For Enhanced Retransmission mode and Streaming mode the Retransmission time-out and Monitor Time-out parameters of the Retransmission and Flow Control option parameters may be changed but all other parameters shall not be changed in a subsequent reconfiguration after the channel has reached the OPEN state.

## 5.5 FRAME CHECK SEQUENCE (FCS) OPTION

This option is used to specify the type of Frame Check Sequence (FCS) that will be included on S/I-frames that are sent. It is non-negotiable. The FCS option shall only be used when the mode is being, or is already configured to Enhanced Retransmission mode or Streaming mode. Note: The FCS option can be reconfigured only when the mode is Enhanced Retransmission mode or Streaming mode. Implementations may reconfigure the FCS when L2CAP channels are moved between Controllers. The Frame Check Sequence option is type 0x05. "No FCS" shall only be used if both L2CAP entities send the FCS Option with value 0x00 (No FCS) in a configuration request. If one L2CAP entity sends the FCS Option with "No FCS" in a configuration request and the other L2CAP sends the FCS Option with a value other than "No FCS" then the default shall be used. If one or both L2CAP entities do not send the FCS option in a configuration request then the default shall be used.

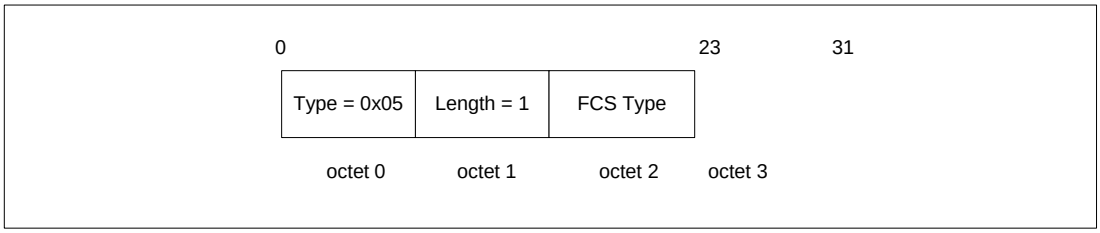



*Figure 5.6: FCS option format*

The FCS types are shown in Table 5.3

| Value | Description |
|---|---|
| 0x00 | No FCS |
| 0x01 | 16-bit FCS defined in section 3.3.5 (default) |
| 0x02 - 0xFF | Reserved |

*Table 5.3: FCS types*

Value of 0x00 is set when the sender wishes to omit the FCS from S/I-frames.

## 5.6 EXTENDED FLOW SPECIFICATION OPTION

This option specifies a flow specification for requesting a desired Quality of Service (QoS) on a channel. It is non-negotiable. The Extended Flow Specification shall be supported on all channels created over AMP-U logical links. Optionally, Extended Flow Specification may be supported on channels created over ACL-U logical links (see Section 7.10). If both devices show support for Extended Flow Specification for BR/EDR in the Extended Feature mask (see Table 4.11) then all channels created between the two devices shall use an Extended Flow Specification. The Quality of Service option and Flush Timeout option shall not be used if the Extended Flow Specification is used.

The parameters in the Extended Flow Specification option specify the traffic stream in the outgoing direction (transmitted traffic). Extended Flow Specification option is type 0x06.

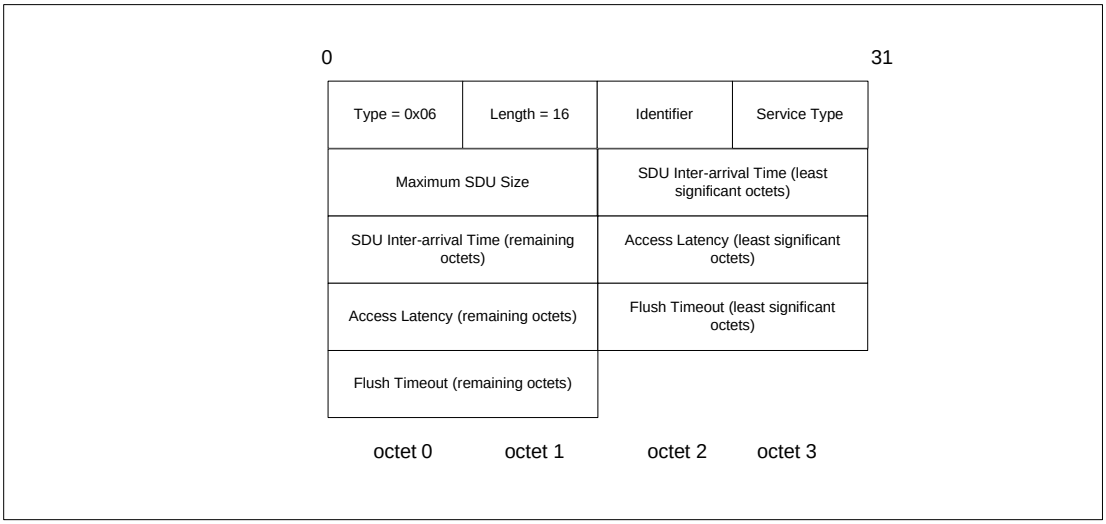



*Figure 5.7: Extended Flow Specification option format*

If no Extended Flow Specification is provided by the upper layer, an Extended Flow Specification with following default values shall be used:

| | |
|---|---|
| Identifier | 0x01 |
| Service Type | Best Effort |
| Maximum SDU size | 0xFFFF |
| SDU Inter-arrival Time | 0xFFFFFFFF |
| Access Latency | 0xFFFFFFFF |
| Flush Timeout | 0xFFFFFFFF |

For a Best Effort channel no latency, air-time or bandwidth guarantees shall be assumed.

The parameters of the Extended Flow Specification are shown in Table 5.4:

| QoS parameter | Parameter Size in Octets | Unit |
|---|---|---|
| Identifier | 1 | na |
| Service Type | 1 | na |
| Maximum SDU Size | 2 | octets |
| SDU Inter-arrival Time | 4 | microseconds |
| Access Latency | 4 | microseconds |
| Flush Timeout | 4 | microseconds |

*Table 5.4: Traffic parameters for L2CAP QoS configuration*

- *Identifier (1 octet)*

  This field provides a unique identifier for the flow specification. This identifier is used by some Controllers in the process of setting up the QoS request. Each active flow specification sent by a device to configure an L2CAP channel shall have a unique Identifier. An Identifier can be reused when the L2CAP channel associated with the flow spec is disconnected. The Identifier shall be unique within the scope of a physical link. Extended Flow Specifications for channels on different physical links may have the same Identifier. The Identifier for an Extended Flow Specification with Service Type Best Effort shall be 0x01.

  Note: Since the Identifier for an Extended Flow Specification with Service Type Best Effort is fixed to 0x01 it is possible to generate a Best Effort Extended Flow Specification for the remote device without performing the Lockstep Configuration process. (The Lockstep Configuration process is described in Section 7.1.3). This allows a Best Effort channel created over the ACL-U logical link without using the Lockstep configuration procedure to be moved to an AMP-U logical link. It is not possible to move a Guaranteed channel created over the ACL-U logical link to an AMP-U logical link unless the Lockstep configuration procedure is used during channel creation. This

is because the Identifier for the remote device's Extended Flow Specification with Service Type Guaranteed is not known.

- *Service Type (1 octet)*

  This field indicates the level of service required. The default value is 'Best effort'. If 'Best effort' is selected then Access Latency and Flush Timeout shall both be set to 0xFFFFFFFF. Maximum SDU size and SDU Inter-arrival Time are used to indicate the maximum data rate that the application can deliver to L2CAP for transmission. This is useful for high speed Controllers so they do not reserve more bandwidth than the application can deliver. The remote device should respond with lower settings indicating the maximum rate at which it can receive data (for example, maximum rate data it can write to a mass storage device, etc.). Values of 0xFFFF for Maximum SDU size and 0xFFFFFFFF for SDU Inter-arrival time are used when the actual values are not known. If Maximum SDU size is set to 0xFFFF then SDU Inter-arrival time shall be set to 0xFFFFFFFF, if SDU Inter-arrival time is set to 0xFFFFFFFF then Maximum SDU size shall be set to 0xFFFF. This tells the Controller to allocate as much bandwidth as possible.

If “Guaranteed” is selected the QoS parameters can be used to identify different types of Guaranteed traffic.

- **Guaranteed bandwidth** traffic is traffic with a minimum data rate but no particular latency requested. Latency will be related to the link supervision timeout. For this type of traffic Access Latency is set to 0xFFFFFFFF.
- **Guaranteed Latency** traffic is traffic with only latency requirements. For this type of traffic SDU Inter-arrival time is set to 0xFFFFFFFF. HID interrupt channel and AVRCP are examples of this type of traffic.
- **Both Guaranteed Latency and Bandwidth** traffic has both a latency and bandwidth requirement. An example is Audio/Video streaming.

If 'No Traffic' is selected the remainder of the fields shall be ignored because there is no data being sent across the channel in the outgoing direction.

| Value | Description |
|---|---|
| 0x00 | No Traffic |
| 0x01 | Best effort (Default) |
| 0x02 | Guaranteed |
| Other | Reserved |

*Table 5.5: Traffic parameters for L2CAP QoS configuration*

A channel shall not be configured as “Best Effort” in one direction and “Guaranteed” in the other direction. If a channel is configured in this way it shall be disconnected. A channel may be configured as “No Traffic” in one direction and “Best Effort” in the other direction. The “No Traffic” refers to the application traffic not the Enhanced Retransmission mode supervisory traffic. A channel con-

figured in this way is considered to have a service type of Best Effort. A channel may be configured as “No Traffic” in one direction and “Guaranteed” in the other direction. A channel configure in this way is considered to have a service type of Guaranteed.

Once configured the service type of a channel shall not be changed during reconfiguration.

- *Maximum SDU Size (2 octets)*

The Maximum SDU Size parameter specifies the maximum size of the SDUs transmitted by the application. If the Service Type is “Guaranteed” then traffic submitted to L2CAP with a larger size is considered non-conformant. QoS guarantees are only provided for conformant traffic.

- *SDU Inter-arrival time (4 octets)*

The SDU Inter-arrival time parameter specifies the time between consecutive SDUs generated by the application. For streaming traffic, SDU Inter-arrival time should be set to the average time between SDUs. For variable rate traffic and best effort traffic, SDU Inter-arrival time should be set to the minimum time between SDUs. If the Service Type is “Guaranteed” then traffic submitted to L2CAP with a smaller interval, is considered non-conformant. QoS guarantees are only provided for conformant traffic.

- *Access Latency (4 octets)*

The Access Latency parameter specifies the maximum delay between consecutive transmission opportunities on the air-interface for the connection. Note that access latency is based on the time base of the Controller, which may not be synchronous to the time base being used by the host or the application.

For streaming traffic (for example, A2DP), the Access Latency should be set to indicate the time budgeted for transmission of the data over the air, and would normally be roughly equal to the Flush Timeout minus the duration of streaming data which can be stored in the receive side application buffers.

For non-streaming, bursty traffic (i.e. HID and AVRCP), the Access Latency parameter value sent in the L2CAP Configuration Request should be set to the desired latency, minus any HCI transport delays and any other stack delays that may be expected on the device and on target host systems. The remote device receiving the L2CAP Configuration Request may send an Access Latency parameter value in the L2CAP Configuration Response which is equal to or lower than the value it received. The remote device may send a lower value to account for other traffic it may be carrying, or overhead activities it may be carrying out.

If HCI is used then the host should take into account the latency of the HCI transport when determining the value for the Access Latency. For example if the application requires an Access Latency of 20ms and the HCI transport has a latency of 5 ms then the value for Access Latency should be 15ms.

- *Flush Timeout (4 Octets)*

The Flush Timeout defines a maximum period after which all segments of the SDU are flushed from L2CAP and the Controller. A Flush Timeout value of 0xFFFFFFFF indicates that data will not be discarded by the transmitting side, even if the link becomes congested. Note that in this case data is treated as reliable, and is never flushed. The device receiving the L2CAP Configuration Request with Flush Timeout set to 0xFFFFFFFF should not modify this parameter. The Flush Timeout for a “Best Effort” channel shall be set to 0xFFFFFFFF.

However, if the Traffic Type is “Guaranteed” and the transmit side buffer is limited, then the Flush Timeout parameter given in the L2CAP Configuration Request may be set to a value corresponding to the duration of streaming data which the transmit buffer can hold before it must begin discarding data. The side receiving the L2CAP Configuration Request may then set the Flush Timeout parameter in the L2CAP Configuration Response to a lower value if the receive side buffer is smaller than the transmit side buffer. Note that the total available buffer space is typically a combination of application buffers, any buffers maintained by the L2CAP implementation, and HCI buffers provided by the Controller.

The Flush Timeout should normally be set to a value which is larger than the Access Latency, and which also accounts for buffers maintained by the application on the receive side such as de-jitter buffers used in audio and video streaming applications. In general, the Flush Timeout value should be selected to ensure that the Flush Timeout expires when the application buffers are about to be exhausted.

Flush Timeout may be set to 0x00000000 to indicate that no retransmissions are to occur. Data may be flushed immediately after transmission in this case. This behavior is useful in applications such as gaming where the tolerable latency is on the order of a few milliseconds, and hence the information contained in a packet will become stale very rapidly. In such applications, it is preferable to send “fresher” data if the last SDU submitted for transmission was not transmitted as a result of interference.

## 5.7 EXTENDED WINDOW SIZE OPTION

This option is used to negotiate the maximum extended window size. The Extended Window Size option is type 0x07 and is non-negotiable.

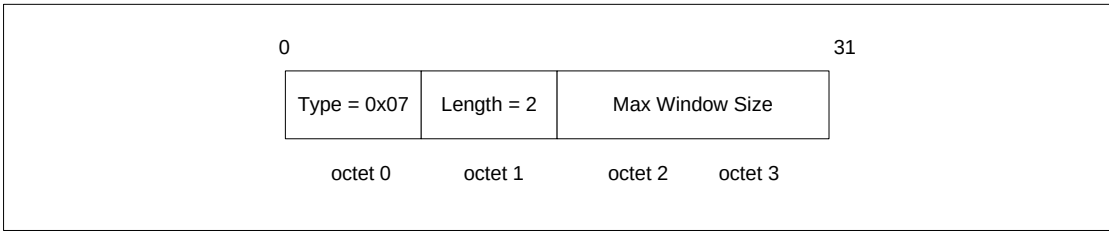



*Figure 5.8: Extended Window Size option format*

The allowable values for the Maximum Extended Window Size are:

| Value | Description |
|---|---|
| 0x0000 | Invalid value for Enhanced Retransmission mode<br>Valid for Streaming mode |
| 0x0001 - 0x3FFF | Valid maximum window size (frames) for Enhanced Retransmission mode. Invalid for Streaming mode |
| 0x4000 - 0xFFFF | Reserved |

*Figure 5.9: Extended Window Size values*

This option shall only be sent if the peer L2CAP entity has indicated support for the Extended Window size feature in the Extended Features Mask (see Section 4.11).

This option shall be ignored if the channel is configured to use Basic L2CAP Mode (see Section 5.4).

For Enhanced Retransmission mode, this option has the same directional semantics as the Retransmission and Flow Control option (see Section 5.4). The sender of a Configuration Request command containing this option is suggesting the maximum window size (possibly based on its own internal L2CAP receive buffer resources) that the peer L2CAP entity should use when sending data.

For Streaming mode, this option is used to enable use of the Extended Control Field and if sent, shall have a value of 0.

If this option is successfully configured then the Maximum Window Size negotiated using the Retransmission and Flow Control option (see Section 5.4) shall be ignored.

If this option is successfully configured in either direction then both L2CAP entities shall use the Extended Control Fields in all S/I-frames (see Section 3.3.2).

If the option is only configured in one direction then the maximum window size for the opposite direction shall be taken from the maximum window size value in the existing Retransmission and Flow Control option. In this configuration both L2CAP entities shall use the extended control fields in all S/I-frames.

# 6 STATE MACHINE

*This section is informative. The state machine may not represent all possible scenarios.*

## 6.1 GENERAL RULES FOR THE STATE MACHINE:

- It is implementation specific, and outside the scope of this specification, how the transmissions are triggered.
- “Ignore” means that the signal can be silently discarded.

The following states have been defined to clarify the protocol; the actual number of states and naming in a given implementation is outside the scope of this specification:

- CLOSED – channel not connected.
- WAIT_CONNECT – a connection request has been received, but only a connection response with indication “pending” can be sent.
- WAIT_CONNECT_RSP – a connection request has been sent, pending a positive connect response.
- CONFIG – the different options are being negotiated for both sides; this state comprises a number of substates, see Section 6.1.3 on page 102
- OPEN – user data transfer state.
- WAIT_DISCONNECT – a disconnect request has been sent, pending a disconnect response.
- WAIT_CREATE – a channel creation request has been received, but only a response with indication “pending” can be sent. This state is similar to WAIT_CONNECT.
- WAIT_CREATE_RSP – a channel creation request has been sent, pending a channel creation response. This state is similar to WAIT_CONNECT_RSP.
- WAIT_MOVE – a request to move the current channel to another Controller has been received, but only a response with indication “pending” can be sent.
- WAIT_MOVE_RSP – a request to move a channel to another Controller has been sent, pending a move response
- WAIT_MOVE_CONFIRM – a response to the move channel request has been sent, waiting for a confirmation of the move operation by the initiator side
- WAIT_CONFIRM_RSP – a move channel confirm has been sent, waiting for a move channel confirm response.

In the following state tables and descriptions, the L2CAP_Data message corresponds to one of the PDU formats used on connection-oriented data channels as described in section 3, including PDUs containing B-frames, I-frames, S-frames.

Some state transitions and actions are triggered only by internal events effecting one of the L2CAP entity implementations, not by preceding L2CAP signaling messages. It is implementation-specific and out of the scope of this specification, how these internal events are realized; just for the clarity of specifying the state machine, the following abstract internal events are used in the state event tables, as far as needed:

- *OpenChannel_Req* – a local L2CAP entity is requested to set up a new connection-oriented channel.
- *OpenChannel_Rsp* – a local L2CAP entity is requested to finally accept or refuse a pending connection request.
- *ConfigureChannel_Req* – a local L2CAP entity is requested to initiate an outgoing configuration request.
- *CloseChannel_Req* – a local L2CAP entity is requested to close a channel.
- *SendData_Req* – a local L2CAP entity is requested to transmit an SDU.
- *ReconfigureChannel_Req* – a local L2CAP entity is requested to reconfigure the parameters of a connection-oriented channel.
- *OpenChannelCntrl_Req* – a local L2CAP entity is requested to set up a new connection over a Controller identified by a Controller ID.
- *OpenChannelCntrl_Rsp* – a local L2CAP entity is requested to internally accept or refuse a pending connection over a Controller identified by a Controller ID.
- *MoveChannel_Req* – a local L2CAP entity is requested to move the current channel to another Controller.
- *ControllerLogicalLinkInd* – a Controller indicates the acceptance or rejection of a logical link request with an Extended Flow Specification.

There is a single state machine for each L2CAP connection-oriented channel that is active. A state machine is created for each new L2CAP_ConnectReq received. The state machine always starts in the CLOSED state.

To simplify the state event tables, the RTX and ERTX timers, as well as the handling of request retransmissions are described in Section 6.1.7 on page 110 and not included in the state tables.

L2CAP messages not bound to a specific data channel and thus not impacting a channel state (e.g. L2CAP_InformationReq, L2CAP_EchoReq) are not covered in this section.

The following states and transitions are illustrated in Figure 6.1 on page 117.

### 6.1.1 CLOSED state

**Notes:**

- The L2CAP_ConnectReq message is not mentioned in any of the other states apart from the CLOSED state, as it triggers the establishment of a new channel, thus the branch into a new instance of the state machine.

| Event | Condition | Action | Next State |
|---|---|---|---|
| *OpenChannel_req* | - | Send L2CAP_ConnectReq | WAIT_CONNECT_RSP |
| L2CAP_ConnectReq | Normal, connection is possible | Send L2CAP_ConnectRsp (success) | CONFIG (substate WAIT_CONFIG) |
| L2CAP_ConnectReq | Need to indicate pending | Send L2CAP_ConnectRsp (pending) | WAIT_CONNECT |
| L2CAP_ConnectReq | No resource, not approved, etc. | Send L2CAP_ConnectRsp (refused) | CLOSED |
| L2CAP_ConnectRsp | - | Ignore | CLOSED |
| L2CAP_ConfigReq | - | Send L2CAP_CommandReject (with reason Invalid CID) | CLOSED |
| L2CAP_ConfigRsp | - | Ignore | CLOSED |
| L2CAP_DisconnectReq | - | Send L2CAP_DisconnectRsp | CLOSED |
| L2CAP_DisconnectRsp | - | Ignore | CLOSED |
| L2CAP_Data | - | Ignore | CLOSED |
| *OpenChannelCntrl_Req* | | Send L2CAP_CreateChanReq | WAIT_CREATE_RSP |
| L2CAP_CreateChanReq | Normal, connection is possible | Send L2CAP_CreateChanRsp (success) | CONFIG |
| L2CAP_CreateChanReq | Need to indicate pending | Send L2CAP_CreateChanRsp (pending) | WAIT_CREATE |
| L2CAP_CreateChanReq | No resource, not approved | Send L2CAP_CreateChanRsp (refused) | CLOSED |
| L2CAP_CreateChanRsp | | Ignore | CLOSED |
| L2CAP_MoveChanReq | | Ignore | CLOSED |
| L2CAP_MoveChanRsp | | Ignore | CLOSED |
| L2CAP_MoveChanConfirm | | Ignore | CLOSED |
| L2CAP_MoveChanConfirmRsp | | Ignore | CLOSED |

*Table 6.1: CLOSED state event table.*

### 6.1.2 WAIT_CONNECT_RSP state

| Event | Condition | Action | Next State |
|---|---|---|---|
| L2CAP_ConnectRsp | Success indicated in result | Send L2CAP_ ConfigReq | CONFIG (substate WAIT_CONFIG) |
| L2CAP_ConnectRsp | Result pending | - | WAIT_CONNECT_RSP |
| L2CAP_ConnectRsp | Remote side refuses connection | - | CLOSED |
| L2CAP_ConfigReq | - | Send L2CAP_ CommandReject (with reason Invalid CID) | WAIT_CONNECT_RSP |
| L2CAP_ConfigRsp | - | Ignore | WAIT_CONNECT_RSP |
| L2CAP_DisconnectRsp | - | Ignore | WAIT_CONNECT_RSP |
| L2CAP_Data | - | Ignore | WAIT_CONNECT_RSP |
| L2CAP_ CreateChanReq | | Ignore | WAIT_CONNECT_RSP |
| L2CAP_ CreateChanRsp | | Ignore | WAIT_CONNECT_RSP |
| L2CAP_MoveChanReq | | Ignore | WAIT_CONNECT_RSP |
| L2CAP_MoveChanRsp | | Ignore | WAIT_CONNECT_RSP |
| L2CAP_ MoveChanConfirm | | Ignore | WAIT_CONNECT_RSP |
| L2CAP_ MoveChanConfirmRsp | | Ignore | WAIT_CONNECT_RSP |

*Table 6.2: WAIT_CONNECT_RSP state event table.*

**Notes:**

- An L2CAP_DisconnectReq message is not included here, since the Source and Destination CIDs are not available yet to relate it correctly to the state machine of a specific channel.

### 6.1.3 WAIT_CONNECT state

| Event | Condition | Action | Next State |
|---|---|---|---|
| *OpenChannel_Rsp* | Pending connection request is finally acceptable | Send L2CAP_Connect_Rsp (success) | CONFIG (substate WAIT_CONFIG) |
| *OpenChannel_Rsp* | Pending connection request is finally refused | Send L2CAP_Connect_Rsp (refused) | CLOSED |
| L2CAP_ConnectRsp | - | Ignore | WAIT_CONNECT |
| L2CAP_ConfigRsp | - | Ignore | WAIT_CONNECT |
| L2CAP_DisconnectRsp | - | Ignore | WAIT_CONNECT |
| L2CAP_Data | - | Ignore | WAIT_CONNECT |
| L2CAP_ CreateChanReq | | Ignore | WAIT_CONNECT |
| L2CAP_ CreateChanRsp | | Ignore | WAIT_CONNECT |
| L2CAP_MoveChanReq | | Ignore | WAIT_CONNECT |
| L2CAP_MoveChanRsp | | Ignore | WAIT_CONNECT |
| L2CAP_ MoveChanConfirm | | Ignore | WAIT_CONNECT |
| L2CAP_ MoveChanConfirmRsp | | Ignore | WAIT_CONNECT |

*Table 6.3: WAIT_CONNECT state event table.*

**Notes:**

- An L2CAP_DisconnectReq or L2CAP_ConfigReq message is not included here, since the Source and Destination CIDs are not available yet to relate it correctly to the state machine of a specific channel.

### 6.1.4 CONFIG state

Two configuration processes exist as described in Section 7.1 on page 120. The configuration processes are the Standard process and the Lockstep process.

In the Standard and Lockstep configuration processes both L2CAP entities initiate a configuration request during the configuration process. This means that each device adopts an initiator role for the outgoing configuration request, and an acceptor role for the incoming configuration request. Configurations in both directions may occur sequentially, but can also occur in parallel.

In the Lockstep configuration process both L2CAP entities send Configuration Request packets and receive Configuration Response packets with a pending result code before submitting the flow specifications to their local controllers. A

final Configuration Response packet is sent by each L2CAP entity indicating the response from its local controller.

The following substates are distinguished within the CONFIG state:

- WAIT_CONFIG – a device has sent or received a connection response, but has neither initiated a configuration request yet, nor received a configuration request with acceptable parameters.
- WAIT_SEND_CONFIG – for the initiator path, a configuration request has not yet been initiated, while for the response path, a request with acceptable options has been received.
- WAIT_CONFIG_REQ_RSP – for the initiator path, a request has been sent but a positive response has not yet been received, and for the acceptor path, a request with acceptable options has not yet been received.
- WAIT_CONFIG_RSP – the acceptor path is complete after having responded to acceptable options, but for the initiator path, a positive response on the recent request has not yet been received.
- WAIT_CONFIG_REQ – the initiator path is complete after having received a positive response, but for the acceptor path, a request with acceptable options has not yet been received.
- WAIT_IND_FINAL_RSP – for both the initiator and acceptor, the Extended Flow Specification has been sent to the local controller but neither a response from controller nor the final configuration response has yet been received.
- WAIT_FINAL_RSP – the device received an indication from the Controller accepting the Extended Flow Specification and has sent a positive response. It is waiting for the remote device to send a configuration response.
- WAIT_CONTROL_IND – the device has received a positive response and is waiting for its Controller to accept or reject the Extended Flow Specification.

According to Section 6.1.1 on page 99 and Section 6.1.2 on page 101, the CONFIG state is entered via WAIT_CONFIG substate from either the CLOSED state, the WAIT_CONNECT state, or the WAIT_CONNECT_RSP state. The CONFIG state is left for the OPEN state if both the initiator and acceptor paths complete successfully.

For better overview, separate tables are given: Table 6.4 shows the success transitions; therein, transitions on one of the minimum paths (no previous non-success transitions) are shaded. Table 6.5 on page 105 shows the non-success transitions within the configuration process, and Table 6.6 on page 106 shows further transition cause by events not belonging to the configuration process itself. The following configuration states and transitions are illustrated in Figure 6.2 on page 118.

| Previous state | Event | Condition | Action | Next State |
|---|---|---|---|---|
| WAIT_CONFIG | *ConfigureChannel_Req* | Standard process | Send L2CAP_Config Req | WAIT_CONFIG_REQ_RSP |
| WAIT_CONFIG | *ConfigureChannel_Req* | Lockstep process | Send L2CAP_Config Req (Ext Flow Spec plus other options) | WAIT_CONFIG_REQ_RSP |
| WAIT_CONFIG | L2CAP_ConfigReq | Options acceptable<br>Standard process | Send L2CAP_Config Rsp (success) | WAIT_SEND_CONFIG |
| WAIT_CONFIG | L2CAP_ConfigReq | Options acceptable<br>Lockstep process | Send L2CAP_Config Rsp (pending) | WAIT_SEND_CONFIG |
| WAIT_CONFIG_REQ_RSP | L2CAP_ConfigReq | Options acceptable | Send L2CAP_Config Rsp (success) | WAIT_CONFIG_RSP |
| WAIT_CONFIG_REQ_RSP | L2CAP_ConfigRsp | Remote side accepts options | - (continue waiting for configuration request) | WAIT_CONFIG_REQ |
| WAIT_CONFIG_REQ | L2CAP_ConfigReq | Options acceptable<br>Standard process | Send L2CAP_Config Rsp (success) | OPEN |
| WAIT_CONFIG_REQ | L2CAP_ConfigReq | Options acceptable<br>Lockstep process | Send L2CAP_Config Rsp (pending) | WAIT_IND_FINAL_RSP |
| WAIT_SEND_CONFIG | *ConfigureChannel_Req* | - | Send L2CAP_Config Req | WAIT_CONFIG_RSP |
| WAIT_CONFIG_RSP | L2CAP_ConfigRsp | Remote side accepts options<br>Standard proc | - | OPEN |
| WAIT_CONFIG_RSP_ | L2CAP_ConfigRsp | Remote side accepts option<br>Lockstep proc | | WAIT_IND_FINAL_RSP |

*Table 6.4: CONFIG state event table*

| Previous state | Event | Condition | Action | Next State |
|---|---|---|---|---|
| WAIT_IND_ FINAL_RSP | ControllerLogical LinkInd | Reject | Send L2CAP_Config Rsp (fail) | WAIT_CONFIG |
| WAIT_IND_ FINAL_RSP | ControllerLogical LinkInd | Accept | Send L2CAP_Config Rsp (success) | WAIT_FINAL_ RSP |
| WAIT_IND_ FINAL_RSP | L2CAP_ConfigRsp | Remote side fail | | WAIT_CONFIG |
| WAIT_IND_ FINAL_RSP | L2CAP_ConfigRsp | Remote side success | | WAIT_ CONTROL_ IND |
| WAIT_FINAL_ RSP | L2CAP_ConfigRsp | Remote side fail | | WAIT_CONFIG |
| WAIT_FINAL_ RSP | L2CAP_ConfigRsp | Remote side success | | OPEN |
| WAIT_ CONTROL_IND | ControllerLogical LinkInd | Reject | Send L2CAP_Config Rsp (fail) | WAIT_CONFIG |
| WAIT_ CONTROL_IND | ControllerLogical LinkInd | Accept | Send L2CAP_Config Rsp (success) | OPEN |

*Table 6.4: CONFIG state event table*

| Previous state | Event | Condition | Action | Next State |
|---|---|---|---|---|
| WAIT_CONFIG | L2CAP_ConfigReq | Options not acceptable | Send L2CAP_Config Rsp (fail) | WAIT_CONFIG |
| WAIT_CONFIG | L2CAP_ConfigRsp | - | Ignore | WAIT_CONFIG |
| WAIT_SEND _CONFIG | L2CAP_ConfigRsp | - | Ignore | WAIT_SEND _CONFIG |
| WAIT_CONFIG _REQ_RSP | L2CAP_ConfigReq | Options not acceptable | Send L2CAP_Config Rsp (fail) | WAIT_CONFIG _REQ_RSP |
| WAIT_CONFIG _REQ_RSP | L2CAP_ConfigRsp | Remote side rejects options | Send L2CAP_Config Req (new options) | WAIT_CONFIG _REQ_RSP |
| WAIT_CONFIG _REQ | L2CAP_ConfigReq | Options not acceptable | Send L2CAP_Config Rsp (fail) | WAIT_CONFIG _REQ |

*Table 6.5: CONFIG state/substates event table: non-success transitions within configuration process.*

| Previous state | Event | Condition | Action | Next State |
|---|---|---|---|---|
| WAIT_CONFIG_REQ | L2CAP_ConfigRsp | - | Ignore | WAIT_CONFIG_REQ |
| WAIT_CONFIG_RSP | L2CAP_ConfigRsp | Remote side rejects options | Send L2CAP_ConfigReq (new options) | WAIT_CONFIG_RSP |

*Table 6.5: CONFIG state/substates event table: non-success transitions within configuration process.*

| Previous state | Event | Condition | Action | Next State |
|---|---|---|---|---|
| CONFIG (any substate) | *CloseChannel_Req* | Any internal reason to stop | Send L2CAP_Disconnect Req | WAIT_DISCONNECT |
| CONFIG (any substate) | L2CAP_Disconnect Req | - | Send L2CAP_Disconnect Rsp | CLOSED |
| CONFIG (any substate) | L2CAP_Disconnect Rsp | - | Ignore | CONFIG (remain in sub-state) |
| CONFIG (any substate) | L2CAP_Data | - | Process the PDU | CONFIG (remain in sub-state) |
| CONFIG (any substate) | L2CAP_CreateChanReq | | Ignore | CONFIG (remain in sub-state) |
| CONFIG (any substate) | L2CAP_CreateChanRsp | | Ignore | CONFIG (remain in sub-state) |
| CONFIG (any substate) | L2CAP_MoveChanReq | | Ignore | CONFIG (remain in sub-state) |
| CONFIG (any substate) | L2CAP_MoveChanRsp | | Ignore | CONFIG (remain in sub-state) |
| CONFIG (any substate) | L2CAP_MoveChanConfirm | | Ignore | CONFIG (remain in sub-state) |
| CONFIG (any substate) | L2CAP_MoveChanCon-firmRsp | | Ignore | CONFIG (remain in sub-state) |

*Table 6.6: CONFIG state/substates event table: events not related to configuration process.*

**Notes:**

- Receiving data PDUs (L2CAP_Data) in CONFIG state should be relevant only in case of a transition to a reconfiguration procedure (from OPEN state). Discarding the

received data is allowed only in Retransmission Mode and Enhanced Retransmission Mode. Discarding an S-frame is allowed but not recommended. If a S-frame is discarded, the monitor timer will cause a new S-frame to be sent after a time out.

- Indicating a failure in a configuration response does not necessarily imply a failure of the overall configuration procedure; instead, based on the information received in the negative response, a modified configuration request may be triggered.

### 6.1.5 OPEN state

| Event | Condition | Action | Next State |
|---|---|---|---|
| *SendData_req* | - | Send L2CAP_Data packet according to configured mode | OPEN |
| *ReconfigureChannel_Req* | - | Complete outgoing SDU<br>Send L2CAP_ConfigReq | CONFIG (substate WAIT_CONFIG_REQ_RSP) |
| *CloseChannel_Req* | - | Send L2CAP_DisconnectReq | WAIT_DISCONNECT |
| L2CAP_ConnectRsp | - | Ignore | OPEN |
| L2CAP_ConfigReq | Incoming config. options acceptable | Complete outgoing SDU<br>Send L2CAP_ConfigRsp (ok) | CONFIG (substate WAIT_SEND_CONFIG) |
| L2CAP_ConfigReq | Incoming config. options not acceptable | Complete outgoing SDU<br>Send L2CAP_ConfigRsp (fail) | OPEN |
| L2CAP_DisconnectReq | - | Send L2CAP_DisconnectRsp | CLOSED |
| L2CAP_DisconnectRsp | - | Ignore | OPEN |
| L2CAP_Data | - | Process the PDU | OPEN |
| MoveChannel_Req | | Send L2CAP_MoveChanReq | WAIT_MOVE_RSP |
| L2CAP_CreateChanReq | | Ignore | OPEN |
| L2CAP_CreateChanRsp | | Ignore | OPEN |
| L2CAP_MoveChanReq | Logical Link create/modify is not needed | Send L2CAP_MoveChanRsp (success) | WAIT_MOVE_CONFIRM |
| L2CAP_MoveChanReq | Need to indicate pending (logical link create/modify is needed) | Send L2CAP_MoveChanRsp (pending) | WAIT_MOVE |
| L2CAP_MoveChanReq | No resource, not approved | Send L2CAP_MoveChanRsp (refused) | WAIT_MOVE_CONFIRM |
| L2CAP_MoveChanRsp | | Ignore | OPEN |

*Table 6.7: OPEN state event table.*

| Event | Condition | Action | Next State |
|---|---|---|---|
| L2CAP_ MoveChanConfirm | | Ignore | OPEN |
| L2CAP_ MoveChanConfirmRsp | | Ignore | OPEN |

*Table 6.7: OPEN state event table.*

Note: The outgoing SDU shall be completed from the view of the remote entity. Therefore all PDUs forming the SDU shall have been reliably transmitted by the local entity and acknowledged by the remote entity, before entering the configuration state.

### 6.1.6 WAIT_DISCONNECT state

| Event | Condition | Action | Next State |
|---|---|---|---|
| L2CAP_ConnectRsp | - | Ignore | WAIT_DISCONNECT |
| L2CAP_ConfigReq | - | Send L2CAP_CommandReject with reason Invalid CID | WAIT_DISCONNECT |
| L2CAP_ConfigRsp | - | Ignore | WAIT_DISCONNECT |
| L2CAP_DisconnectReq | - | Send L2CAP_DisconnectRsp | CLOSED |
| L2CAP_DisconnectRsp | - | - | CLOSED |
| L2CAP_Data | - | Ignore | WAIT_DISCONNECT |
| L2CAP_ CreateChanReq | | Ignore | WAIT_DISCONNECT |
| L2CAP_ CreateChanRsp | | Ignore | WAIT_DISCONNECT |
| L2CAP_MoveChanReq | | Ignore | WAIT_DISCONNECT |
| L2CAP_MoveChanRsp | | Ignore | WAIT_DISCONNECT |
| L2CAP_MoveChanConfirm | | Ignore | WAIT_DISCONNECT |
| L2CAP_MoveChanConfirmRsp | | Ignore | WAIT_DISCONNECT |

*Table 6.8: WAIT_DISCONNECT state event table.*

### 6.1.7 WAIT_CREATE_RSP state

| Event | Condition | Action | Next State |
|---|---|---|---|
| L2CAP_CreateChanReq | | Ignore | WAIT_CREATE_RSP |
| L2CAP_CreateChanRsp | Success indicated in result | Send L2CAP_ConfigReq | CONFIG (substate WAIT_CONFIG) |
| L2CAP_CreateChanRsp | Result pending | | WAIT_CREATE_RSP |
| L2CAP_CreateChanRsp | Remote side refuses connection | | CLOSED |
| L2CAP_MoveChanReq | | Ignore | WAIT_CREATE_RSP |
| L2CAP_MoveChanRsp | | Ignore | WAIT_CREATE_RSP |
| L2CAP_MoveChanConfirm | | Ignore | WAIT_CREATE_RSP |
| L2CAP_ MoveChanConfirmRsp | | Ignore | WAIT_CREATE_RSP |

*Table 6.9: WAIT_CREATE_RSP state event table.*

### 6.1.8 WAIT_CREATE state

| Event | Condition | Action | Next State |
|---|---|---|---|
| OpenChannelCntrl_Req | - | Ignore | WAIT_CREATE |
| OpenChannelCntrl_Rsp | Pending connection request is finally acceptable | Send L2CAP_ CreateChanRsp (success) | CONFIG |
| OpenChannelCntrl_Rsp | Pending connection request is finally refused | Send L2CAP_ CreateChanRsp (refused) | CLOSED |
| L2CAP_CreateChanReq | - | Ignore | WAIT_CREATE |
| L2CAP_CreateChanRsp | - | Ignore | WAIT_CREATE |
| L2CAP_MoveChanReq | - | Ignore | WAIT_CREATE |
| L2CAP_MoveChanRsp | - | Ignore | WAIT_CREATE |
| L2CAP_ MoveChanConfirm | - | Ignore | WAIT_CREATE |
| L2CAP_ MoveChanConfirmRsp | - | Ignore | WAIT_CREATE |

*Table 6.10: WAIT_CREATE state event table*

### 6.1.9 WAIT_MOVE_RSP state

| Event | Condition | Action | Next State |
|---|---|---|---|
| L2CAP_MoveChanReq | Prioritization algorithm (remain initiator) | Send L2CAP_MoveChanRsp (collision) | WAIT_MOVE_RSP |
| L2CAP_MoveChanReq | Prioritization algorithm (become responder) and logical link create/modify is needed) | Send L2CAP_MoveChanRsp (pending) | WAIT_MOVE |
| L2CAP_MoveChanReq | Prioritization algorithm (become responder) and logical link create/modify is not needed) | Send L2CAP_MoveChanRsp (success) | WAIT_MOVE_CONFIRM |
| L2CAP_CreateChanRsp | - | Ignore | WAIT_MOVE_RSP |
| L2CAP_CreateChanReq | - | Ignore | WAIT_MOVE_RSP |
| L2CAP_MoveChanRsp | Result pending | - | WAIT_MOVE_RSP |
| L2CAP_MoveChanRsp | Remote side refuses move and/or local side refuses move (create/modify logical link fails) | Send L2CAP_MoveChanConfirm (failure) | WAIT_CONFIRM_RSP |
| L2CAP_MoveChanRsp | Both remote side and local side accepts move (create/modify logical link succeeds) | Send L2CAP_MoveChanConfirm (success) | WAIT_CONFIRM_RSP |
| L2CAP_MoveChanConfirm | - | Ignore | WAIT_MOVE_RSP |

*Table 6.11: WAIT_MOVE_RSP state event table*

| Event | Condition | Action | Next State |
|---|---|---|---|
| L2CAP_ MoveChanConfirmRsp | - | Ignore | WAIT_MOVE_RSP |

*Table 6.11: WAIT_MOVE_RSP state event table*

### 6.1.10 WAIT_MOVE state

| Event | Condition | Action | Next State |
|---|---|---|---|
| ControllerLogicalLinkInd | Create/modify logical link completed successfully | Send L2CAP_MoveChanRsp (success) | WAIT_MOVE_CONFIRM |
| ControllerLogicalLinkInd | Create/modify logical link completed with failure | Send L2CAP_MoveChanRsp (refused) | WAIT_MOVE_CONFIRM |
| L2CAP_CreateChanReq | - | Ignore | WAIT_MOVE |
| L2CAP_CreateChanRsp | - | Ignore | WAIT_MOVE |
| L2CAP_MoveChanReq | - | Ignore | WAIT_MOVE |
| L2CAP_MoveChanRsp | - | Ignore | WAIT_MOVE |
| L2CAP_MoveChanConfirm | - | Ignore | WAIT_MOVE |
| L2CAP_MoveChanConfirmRsp | - | Ignore | WAIT_MOVE |

*Table 6.12: WAIT_MOVE state event table*

### 6.1.11 WAIT_MOVE_CONFIRM state

| Event | Condition | Action | Next State |
|---|---|---|---|
| L2CAP_CreateChanReq | - | Ignore | WAIT_MOVE_CONFIRM |
| L2CAP_CreateChanRsp | - | Ignore | WAIT_MOVE_CONFIRM |
| L2CAP_MoveChanReq | - | Ignore | WAIT_MOVE_CONFIRM |
| L2CAP_MoveChanRsp | - | Ignore | WAIT_MOVE_CONFIRM |
| L2CAP_MoveChanConfirm | Result of move is success | Send L2CAP_MoveChanConfirmRsp | OPEN (on new Controller) |
| L2CAP_MoveChanConfirm | Result of move is failure | Send L2CAP_MoveChanConfirmRsp | OPEN (on old Controller) |
| L2CAP_MoveChanConfirmRsp | - | Ignore | WAIT_MOVE_CONFIRM |

*Table 6.13: WAIT_MOVE_CONFIRM state event table*

### 6.1.12 WAIT_CONFIRM_RSP state

| Event | Condition | Action | Next State |
|---|---|---|---|
| L2CAP_CreateChanReq | - | Ignore | WAIT_CONFIRM_RSP |
| L2CAP_CreateChanRsp | - | Ignore | WAIT_CONFIRM_RSP |
| L2CAP_MoveChanReq | - | Ignore | WAIT_CONFIRM_RSP |
| L2CAP_MoveChanRsp | - | Ignore | WAIT_CONFIRM_RSP |
| L2CAP_MoveChanConfirm | - | Ignore | WAIT_CONFIRM_RSP |
| L2CAP_MoveChanConfirmRsp | Move succeeded | | OPEN (on new Controller) |
| L2CAP_MoveChanConfirmRsp | Move failed | | OPEN (on old Controller) |

*Table 6.14: WAIT_CONFIRM_RSP state event table*

## 6.2 TIMERS EVENTS

### 6.2.1 RTX

The Response Timeout eXpired (RTX) timer is used to terminate the channel when the remote endpoint is unresponsive to signaling requests. This timer is started when a signaling request (see Section 7 on page 120) is sent to the remote device. This timer is disabled when the response is received. If the ini-

tial timer expires, a duplicate Request message may be sent or the channel identified in the request may be disconnected. If a duplicate Request message is sent, the RTX timeout value shall be reset to a new value at least double the previous value. When retransmitting the Request message, the context of the same state shall be assumed as with the original transmission. If a Request message is received that is identified as a duplicate (retransmission), it shall be processed in the context of the same state which applied when the original Request message was received.

Implementations have the responsibility to decide on the maximum number of Request retransmissions performed at the L2CAP level before terminating the channel identified by the Requests. The exception is fixed channel CIDs since they can never be terminated. The LE Peripheral may disconnect the link on the expiry of the RTX timer. The decision should be based on the flush timeout of the signaling link. The longer the flush timeout, the more retransmissions may be performed at the physical layer and the reliability of the channel improves, requiring fewer retransmissions at the L2CAP level.

For example, if the flush timeout is infinite, no retransmissions should be performed at the L2CAP level. When terminating the channel, it is not necessary to send a L2CAP_DisconnectReq and enter WAIT_DISCONNECT state. Channels can be transitioned directly to the CLOSED state.

The value of this timer is implementation-dependent but the minimum initial value is 1 second and the maximum initial value is 60 seconds. One RTX timer shall exist for each outstanding signaling request, including each Echo Request. The timer disappears on the final expiration, when the response is received, or the physical link is lost. The maximum elapsed time between the initial start of this timer and the initiation of channel termination (if no response is received) is 60 seconds.

### 6.2.2 ERTX

The Extended Response Timeout eXpired (ERTX) timer is used in place of the RTX timer when it is suspected the remote endpoint is performing additional processing of a request signal. This timer is started when the remote endpoint responds that a request is pending, e.g., when an L2CAP_ConnectRsp event with a “connect pending” result (0x0001) is received. This timer is disabled when the formal response is received or the physical link is lost. If the initial timer expires, a duplicate Request may be sent or the channel may be disconnected.

If a duplicate Request is sent, the particular ERTX timer disappears, replaced by a new RTX timer and the whole timing procedure restarts as described previously for the RTX timer.

The value of this timer is implementation-dependent but the minimum initial value is 60 seconds and the maximum initial value is 300 seconds. Similar to RTX, there MUST be at least one ERTX timer for each outstanding request that received a Pending response. There should be at most one (RTX or ERTX) associated with each outstanding request. The maximum elapsed time between the initial start of this timer and the initiation of channel termination (if no response is received) is 300 seconds. When terminating the channel, it is not necessary to send a L2CAP_DisconnectReq and enter WAIT_DISCONNECT state. Channels should be transitioned directly to the CLOSED state.

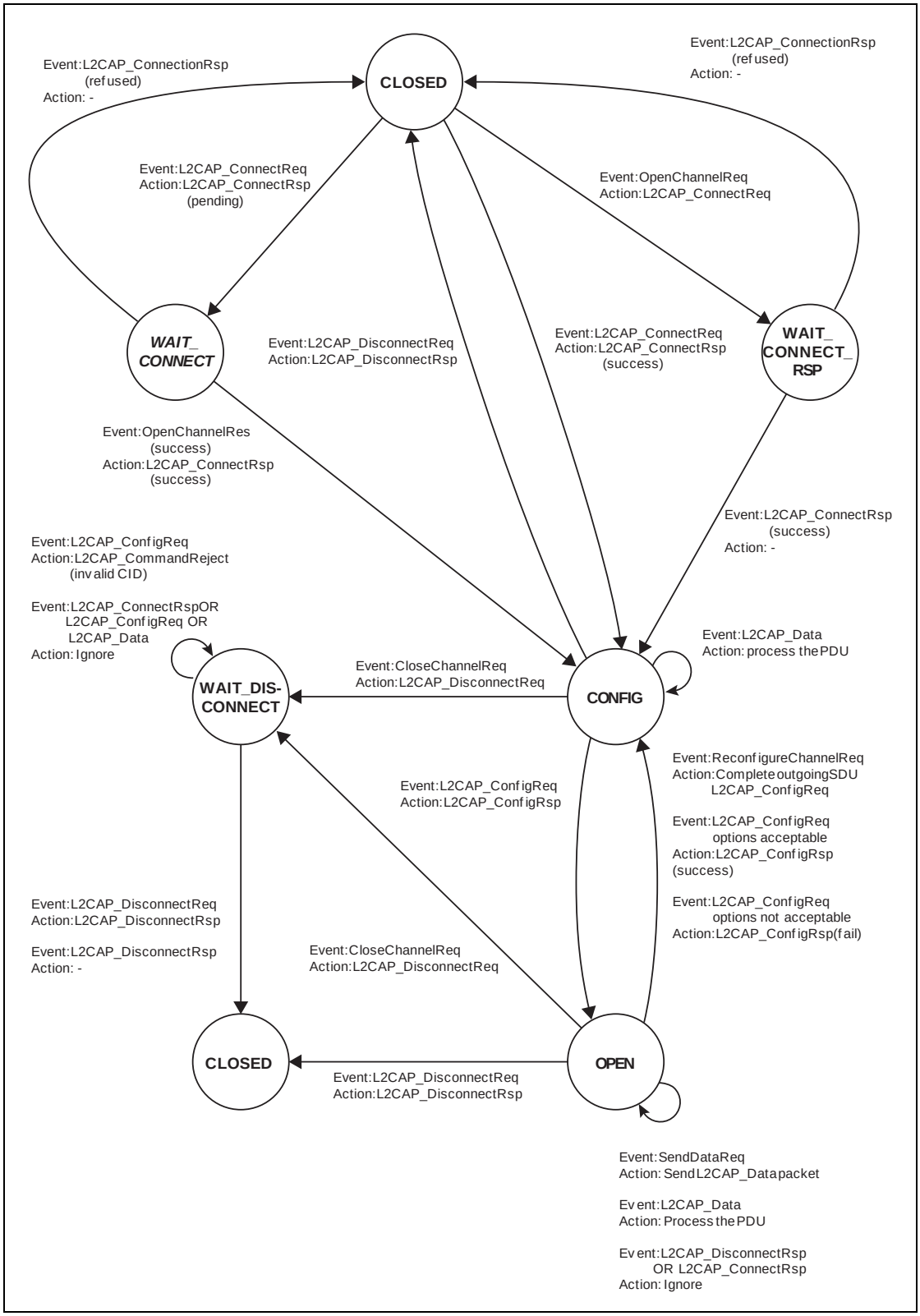



*Figure 6.1: States and transitions.*

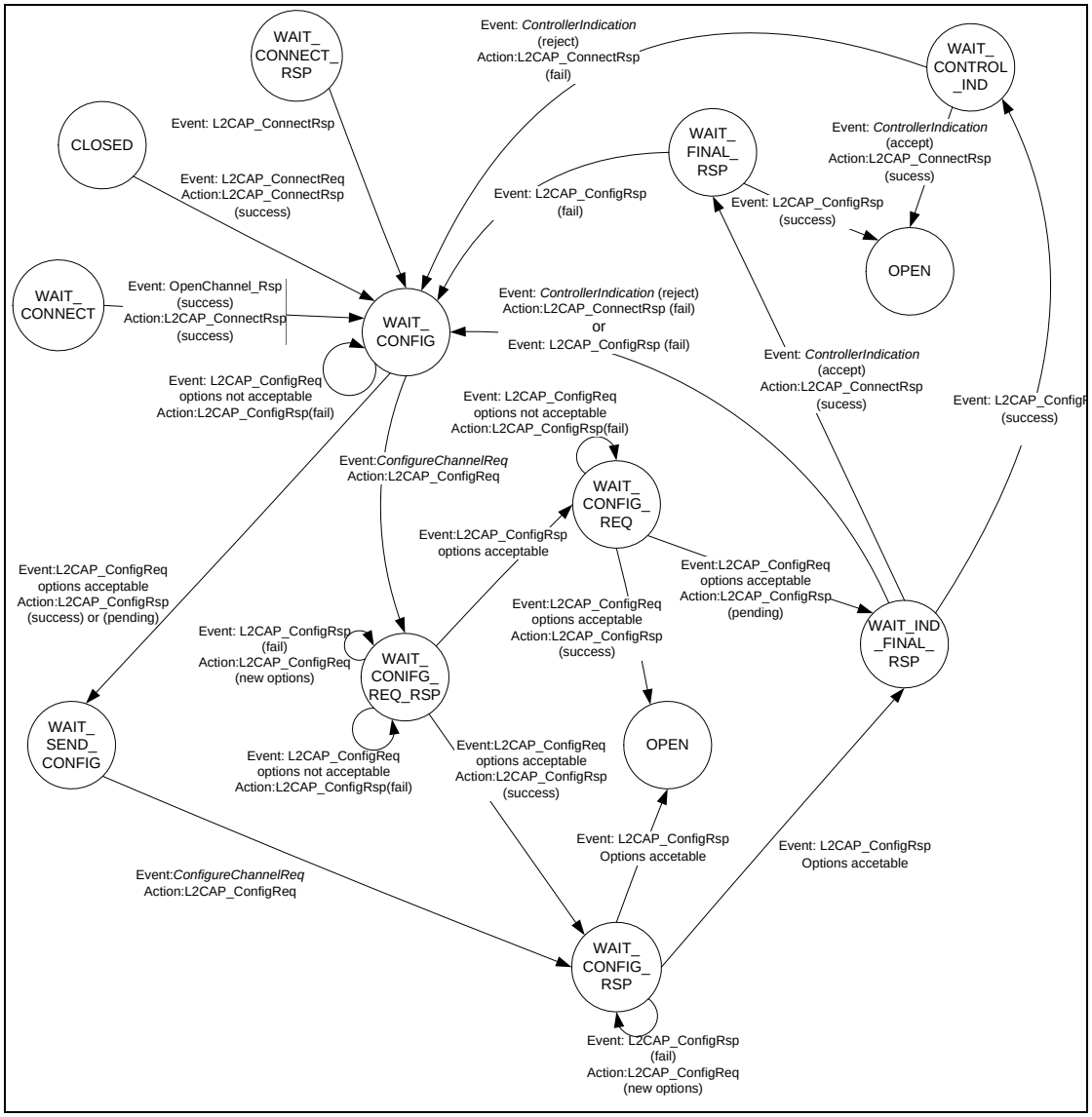



*Figure 6.2: Configuration states and transitions.*

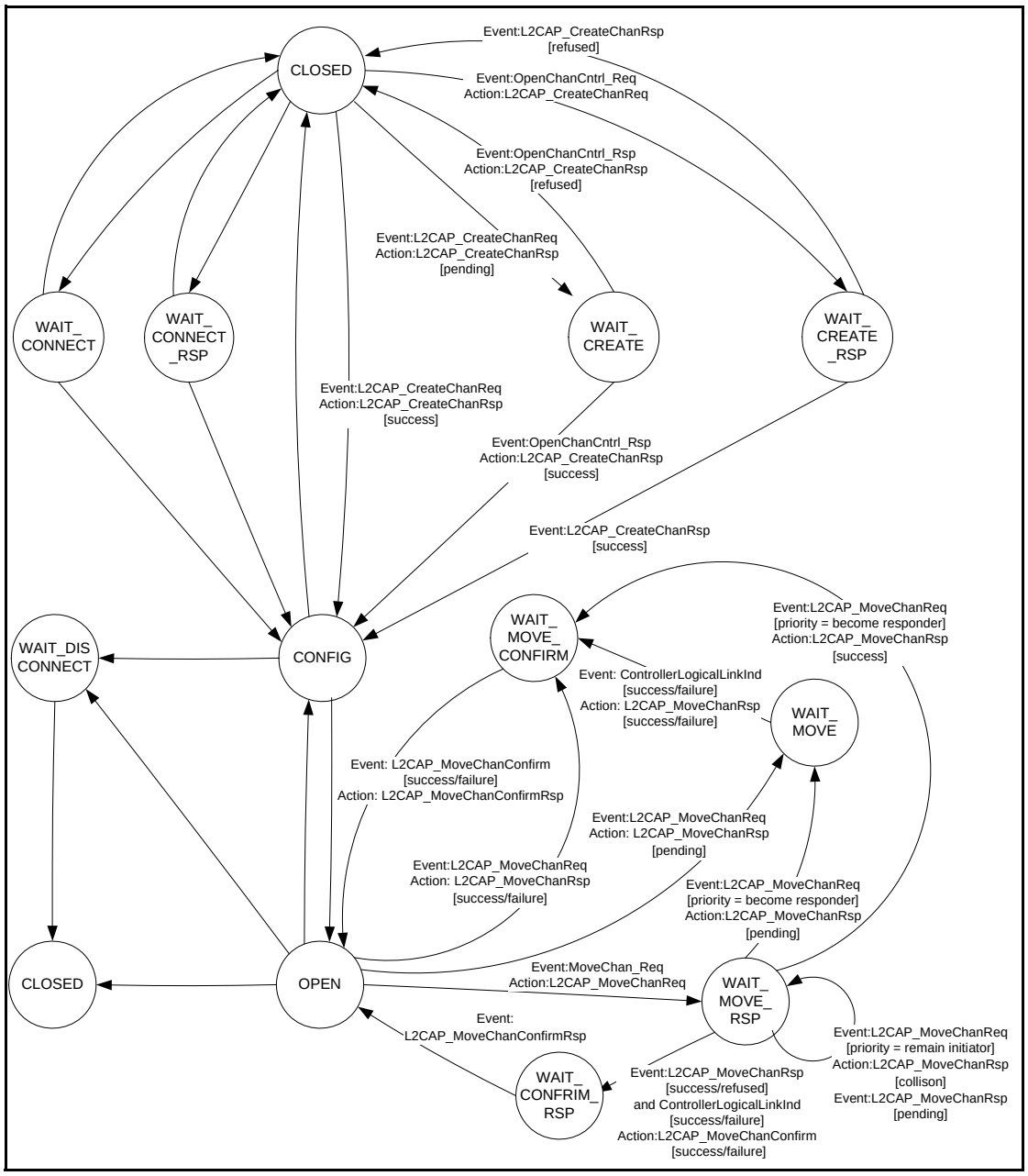



*Figure 6.3: States and transitions–AMP enabled operation*

Figure 6.3 is based on Figure 6.1 on page 117. Only the new transitions show event/action labels. Transitions without labels are the same as in Figure 6.1.

# 7 GENERAL PROCEDURES

This section describes the general operation of L2CAP, including the configuration process, the handling and the processing of user data for transportation over the air interface. This section also describes the operation of L2CAP features including the delivery of erroneous packets, the flushing of expired data and operation in connectionless mode, operation collision resolution, aggregation of best effort flow specifications, and prioritizing data over HCI.

Procedures for the flow control and retransmission modes are described in Section 8 on page 139.

## 7.1 CONFIGURATION PROCESS

Configuration consists of two processes, the Standard process and the Lockstep process. The Lockstep process shall be used if both L2CAP entities support the Extended Flow Specification option otherwise the Standard process shall be used.

For both processes, configuring the channel parameters shall be done independently for both directions. Both configurations may be done in parallel.

Each configuration parameter is one-directional. The configuration parameters describe the non default parameters the device sending a Configuration Request will accept. The configuration request cannot request a change in the parameters the device receiving the request will accept.

If a device needs to establish the value of a configuration parameter the remote device will accept, then it must wait for a configuration request containing that configuration parameter to be sent from the remote device.

8. The Lockstep process is used when the channel parameters include an Extended Flow Specification option. The Lockstep process can be divided into two phases. In the first phase, each L2CAP entity shall receive an Extended Flow Specification option along with all non-default parameters from its peer L2CAP entity and then present the pair of Extended Flow specifications (local and remote) to its local Controller. In the second phase, each L2CAP entity shall report the result from its local Controller to the peer L2CAP entity. The Lockstep process is described in Section 7.1.3. The Standard process is described in Section 7.1.4.

Both the Lockstep and Standard processes can be abstracted into the initial Request configuration path and a Response configuration path, followed by the reverse direction phase. Reconfiguration follows a similar two-phase process by requiring configuration in both directions.

During a reconfiguration session, all data traffic on the channel should be suspended pending the outcome of the configuration. If an L2CAP entity receives a Configuration Request while it is waiting for a response it shall not block

sending the Configuration Response, otherwise the configuration process may deadlock.

### 7.1.1 Request Path

The Request Path can configure the following:

- requester's incoming MTU.
- requester's outgoing flush timeout.
- requester's outgoing QoS.
- requester's incoming and outgoing flow and error control information.
- requester's incoming and outgoing Frame Check Sequence option.
- requester's outgoing QoS using Extended Flow Specification option.
- requester's incoming Extended Window size option plus incoming and outgoing frame format.

Table 7.1 on page 121 defines the configuration options that may be placed in a Configuration Request.

| Parameter | Description |
|---|---|
| MTU | Incoming MTU information |
| FlushTO | Outgoing flush timeout[1] |
| QoS | Outgoing QoS information[1] |
| RFCMode | Incoming and outgoing Retransmission and Flow Control Mode |
| FCS | Incoming and outgoing Frame Check Sequence |
| ExtFlowSpec | Outgoing QoS information[1] |
| ExtWindow | Incoming Extended Window size plus incoming and outgoing frame format |

*Table 7.1: Parameters allowed in Request*

1. FlushTO, QoS and ExtFlowSpec are considered QoS information. ExtFlowSpec is used instead of FlushTO and QoS when both devices support ExtFlowSpec.

The state machine for the configuration process is described in Section 6 on page 98.

### 7.1.2 Response pPath

The Response Path can configure the following:

- responder's outgoing MTU, that is the remote side's incoming MTU.

- remote side's flush timeout.
- responder's incoming QoS Flow Specification (remote side's outgoing QoS Flow Specification).
- responder's incoming and outgoing Flow and Error Control information
- responder's incoming QoS Extended Flow Specification (remote side's outgoing QoS Flow Specification)
- responder's outgoing Extended Window size.

For the Standard process, if a request-oriented parameter is not present in the Request message (reverts to last agreed value), the remote side may negotiate for a non-default value by including the proposed value in a negative Response message. Table 7.2 defines the configuration options that may be placed in a Configuration Response.

| Parameter | Description |
|---|---|
| MTU | Outgoing MTU information |
| FlushTO | Incoming flush timeout [1] |
| QoS | Incoming QoS information[1] |
| RFCMode | Incoming and outgoing Retransmission and Flow Control Mode |
| ExtFlowSpec | Incoming QoS information[1] |
| ExtWindow | Outgoing Extended Window size |

*Table 7.2: Parameters allowed in Response*

1. FlushTO, QoS and ExtFlowSpec are considered QoS information. ExtFlowSpec is used instead of FlushTO and QoS when both devices support ExtFLowSpec

### 7.1.3 Lockstep Configuration Process

Configuration Request packets are sent to establish or change the channel parameters including the QoS contract between two L2CAP entities. An Extended Flow Specification option along with any non-default parameters shall be sent in a Configuration Request following the sending or receipt of a Connect Response which accepts an L2CAP Connect Request for the channel being configured. Unlike the Standard process, negotiation involving the sending of multiple Configuration Request packets shall not be performed. In other words, only one Configuration Request packet shall be sent by each L2CAP entity during the Lockstep configuration process. The Lockstep process shall only be used during channel reconfiguration when an Extended Flow Specification option is present in the Configuration Request packet used for reconfiguration.

The Extended Flow Specification shall be sent for all channels created on AMP-U logical links and shall only be sent for channels created on the ACL-U logical link if both the local and remote L2CAP entities have indicated support for the Extended Flow Specification for BR/EDR in their Extended Features masks. The Extended Features mask shall be obtained via the L2CAP Information Request/Response signaling mechanism (InfoType = 0x0002) prior to the issuance of the L2CAP Configuration Request. If a Configuration Request is sent containing the Extended Flow Specification option then the Quality of Service option and Flush Timeout option shall not be included.

Configuration Response packets shall be sent in reply to Configuration Request packets except when the error condition is covered by a Command Reject response. Although the L2CAP Configuration signaling mechanism allows for the use of wildcards, wildcard values are not supported in the Extended Flow Specification since each parameter represents a property of the traffic in one direction only, and no negotiation is intended to be performed at the L2CAP Configuration level.

The recipient of a Configuration Request shall perform all necessary checks with the Controller to validate that the requested QoS can be granted. In the case of a BR/EDR or BR/EDR/LE Controller the L2CAP layer performs the Controller checks. If HCI is used then the L2CAP entity should check that the requested QoS can be achieved over the HCI transport. In order to perform these checks, the recipient needs to have the QoS parameters for both directions. In order for each side to determine when to perform relevant Controller checks, each side will reply with a result “Pending” (0x0004). If no parameters are sent in the Configuration Response packet with result “Pending” then the parameters sent in the Configuration Request have been accepted without change. This Lockstep procedure results in both L2CAP entities performing the following:

- Receiving a Configuration Request containing the Extended Flow Specification option along with all non-default parameters
- Sending a Configuration Response with any modifications to the Extended Flow Specification option and allowed modifications to non-default parameters (result “Pending”)
- Sending a Configuration Request containing the Extended Flow Specification option along with all non-default parameters.
- Receiving a Configuration Response containing any modifications to the Extended Flow Specification option and allowed modifications to non-default parameters (result “Pending”)

The ERTX timer is used when a Configuration Response is received with result “Pending” (see Section 6.2.2).

If a Configuration Response with result “success” is received before a Configuration Response with result “pending” is received the recipient shall disconnect the channel. This is a violation of the Lockstep configuration process.

If a device sends an Extended Flow Specification option in a Configuration Request with service type “Best Effort” and receives a Configuration Request with service type “Guaranteed,” the channel shall be disconnected. If a device sends an Extended Flow Specification in a Configuration Request with type “Guaranteed” and receives a Configuration Request with service type “Best Effort,” the channel shall be disconnected.

If the service type is “Best Effort” then values for certain parameters may be sent in a Configuration Response with result “Pending” to indicate the maximum bandwidth the sender of the Configuration Response is capable to receive. The values sent in a Configuration Response with result “Pending” and service type Best Effort shall be in accordance with Table 7.3.

| Parameter | Changes permitted by responder |
|---|---|
| Maximum SDU Size | May decrease only |
| SDU Inter-arrival Time | May increase only |
| Access Latency | None |
| Flush Timeout | None |

*Table 7.3: Permitted parameter in L2CAP Configuration Response for Service Type “Best Effort”*

After the Configuration Response is received with result “Pending,” L2CAP may issue the necessary checks with the Controller.

If the Controller cannot support the Extended Flow Specifications with service type “Guaranteed,” then the recipient of the Configuration Request shall send a Configuration Response indicating a result code of “Failure - flow spec rejected” (0x0005). If the Controller indicates that it can support the Extended Flow Specifications, then the recipient of the Configuration Request shall send a Configuration Response with result code of “Success” (0x0000) with no parameters.

If the Result of the Configuration Response is Failure (0x0005) for service type “Guaranteed” then an Extended Flow Specification option may be sent in the Configuration Response. The Extended Flow Specification parameters sent in the Configuration Response may be changed to reflect a QoS level that would be acceptable, but shall only be changed in accordance with Table 7.4.

| Parameter | Changes permitted by responder |
|---|---|
| Maximum SDU Size | None |
| SDU Inter-arrival Time | None |
| Access Latency | May decrease only |

*Table 7.4: Permitted parameter changes in L2CAP Configuration Response for Service Type “Guaranteed”*

| Parameter | Changes permitted by responder |
|---|---|
| Flush Timeout | May decrease only (unless set to 0xFFFFFFFF, in which case no change is permitted) |

*Table 7.4: Permitted parameter changes in L2CAP Configuration Response for Service Type "Guaranteed"*

If an L2CAP Configuration Response is received containing the Extended Flow Specification option with the same values sent earlier, the upper layer shall be notified of the error.

### 7.1.4 Standard Configuration Process

For the Standard process the following general procedure shall be used for each direction:

1. Local device informs the remote device of the parameters that the local device will accept using a Configuration Request.
2. Remote device responds, agreeing or disagreeing with these values, including the default ones, using a Configuration Response.
3. The local and remote devices repeat steps (1) and (2) until agreement on all parameters is reached.

The decision on the amount of time (or messages) spent configuring the channel parameters before terminating the configuration is left to the implementation, but it shall not last more than 120 seconds.

There are two types of configuration parameters: negotiable and non-negotiable.

Negotiable parameters are those that a remote side receiving a Configuration Request can disagree with by sending a Configuration Response with the Unacceptable Parameters (0x0001) result code, proposing new values that can be accepted. Non-negotiable parameters are only informational and the recipient of a Configuration Request cannot disagree with them but can provide adjustments to the values in a positive Configuration Response. Section 5 identifies each parameter as negotiable or non-negotiable. Note: MTU is non-negotiable but can be rejected if a value lower than the mandated minimum is proposed (See Section 5.1).

The following rules shall be used for parameter negotiation in the Request Path:

1. An L2CAP entity shall send at least one Configuration Request packet as part of initial configuration or reconfiguration. If all default or previously agreed values are acceptable, a Configuration Request packet with no options shall be sent.
2. When an L2CAP entity receives a positive Configuration Response from the remote device it shall consider all configuration parameters explicitly con-

tained in the Configuration Request along with the default and previously agreed values not explicitly contained in the Configuration Request as accepted by the remote device.

3. When an L2CAP entity receives a negative Configuration Response and sends a new Configuration Request it shall include all the options it sent in the previous Configuration Request with the same values except the negotiable options that were explicitly rejected in the negative Configuration Response which will have new values. Note: the resending of all the options provides backwards compatibility with remote implementations that don't assume rule 4 when responding.
4. Negotiable options not included in a negative Configuration Response are considered accepted by the remote device. Note: for backwards compatibility if non-negotiable options are included in a negative Configuration Response, they should be processed as if the response was positive.

The following rules shall be used for parameter negotiation in the Response Path:

1. A positive Configuration Response accepts the values of all configuration parameters explicitly contained in the received Configuration Request and the default and previously agreed values not explicitly provided.
2. An L2CAP Entity shall send a negative Configuration Response to reject negotiable parameter values that are unacceptable, be it values explicitly provided in the received Configuration Request or previously agreed or default values. The rejected parameters sent in a negative Configuration Response shall have values that are acceptable to the L2CAP entity sending the negative Configuration Response.
3. All negotiable options being rejected shall be rejected in the same negative Configuration Response.
4. The only options allowed in a negative Configuration Response are the negotiable options being rejected. No wildcards or adjustments to non-negotiable options shall be in a negative Configuration Response.
5. Negotiable options not included in the negative Configuration Response are considered accepted.

## 7.2 FRAGMENTATION AND RECOMBINATION

Fragmentation is the breaking down of PDUs into smaller pieces for delivery from L2CAP to the lower layer. Recombination is the process of reassembling a PDU from fragments delivered up from the lower layer. Fragmentation and Recombination may be applied to any L2CAP PDUs.

### 7.2.1 Fragmentation of L2CAP PDUs

An L2CAP implementation may fragment any L2CAP PDU for delivery to the lower layers. If L2CAP runs directly over the Controller without HCI, then an

implementation may fragment the PDU into multiple Controller packets for transmission over the air. If L2CAP runs above the HCI, then an implementation may send HCI transport sized fragments to the Controller. All L2CAP fragments associated with an L2CAP PDU shall be processed for transmission by the Controller before any other L2CAP PDU for the same logical transport shall be processed.

For example, in the BR/EDR controller the two LLID bits defined in the first octet of baseband payload (also called the frame header) are used to signal the start and continuation of L2CAP PDUs. LLID shall be ‘10’ for the first segment in an L2CAP PDU and ‘01’ for a continuation segment. An illustration of fragmentation for a BR/EDR controller is shown in Figure 7.1 on page 127. An example of how fragmentation might be used in a device with HCI is shown in Figure 7.2 on page 128.

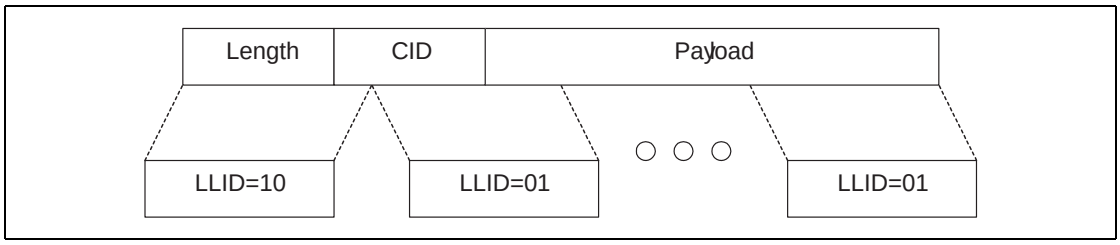



*Figure 7.1: L2CAP fragmentation in a BR/EDR Controller*

Note: The BR/EDR link controller is able to impose a different fragmentation on the PDU by using “start” and “continuation” indications as fragments are translated into baseband packets. Thus, both L2CAP and the BR/EDR link controller use the same mechanism to control the size of fragments.

### 7.2.2 Recombination of L2CAP PDUs

Controllers will attempt to deliver packets in sequence and without errors. Recombination of fragments may occur in the Controller but ultimately it is the responsibility of L2CAP to reassemble PDUs and SDUs and to check the length field of the SDUs. As the Controller receives packet fragments, it either signals the L2CAP layer on the arrival of each fragment, or accumulates a number of fragments (before the receive buffer fills up or a timer expires) before passing packets to the L2CAP layer.

An L2CAP implementation shall use the length field in the header of L2CAP PDUs, see Section 3 on page 44, as a consistency check and shall discard any L2CAP PDUs that fail to match the length field. If channel reliability is not needed, packets with invalid lengths may be silently discarded. For reliable channels using Basic mode, an L2CAP implementation shall indicate to the upper layer that the channel has become unreliable. Reliable channels are defined by having an infinite flush timeout value as specified in Section 5.2 on page 81. For higher data integrity L2CAP should be operated in the Enhanced Retransmission Mode.

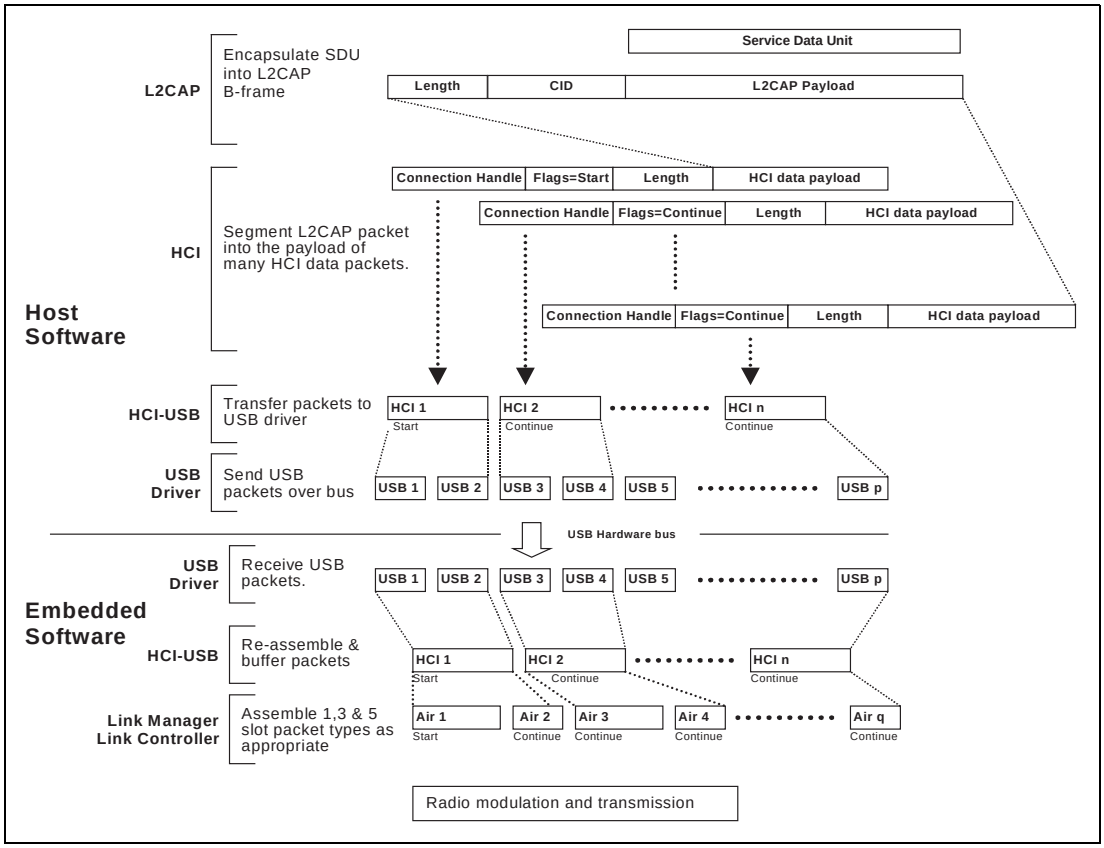



*Figure 7.2: Example of fragmentation processes in a device with a BR/EDR Controller and USB HCI transport.*

## 7.3 ENCAPSULATION OF SDUs

All SDUs are encapsulated into one or more L2CAP PDUs.

In Basic L2CAP mode, an SDU shall be encapsulated with a minimum of L2CAP protocol elements, resulting in a type of L2CAP PDU called a Basic Information Frame (B-frame).

Segmentation and Reassembly operations are only used in Enhanced Retransmission mode, Streaming mode, Retransmission mode, and Flow Control mode. SDUs may be segmented into a number of smaller packets called SDU segments. Each segment shall be encapsulated with L2CAP protocol elements resulting in an L2CAP PDU called an Information Frame (I-frame).

The maximum size of an SDU segment shall be given by the Maximum PDU Payload Size (MPS). The MPS parameter may be exported using an implementation specific interface to the upper layer.

Note that this specification does not have a normative service interface with the upper layer, nor does it assume any specific buffer management scheme of a host implementation. Consequently, a reassembly buffer may be part of the upper layer entity. It is assumed that SDU boundaries shall be preserved between peer upper layer entities.

### 7.3.1 Segmentation of L2CAP SDUs

In Flow Control, Streaming, or Retransmission modes, incoming SDUs may be broken down into segments, which shall then be individually encapsulated with L2CAP protocol elements (header and checksum fields) to form I-frames. I-frames are subject to flow control and may be subject to retransmission procedures. The header carries a 2 bit SAR field that is used to identify whether the I-frame is a 'start', 'end' or 'continuation' packet or whether it carries a complete, unsegmented SDU. Figure 7.3 on page 129 illustrates segmentation and fragmentation.

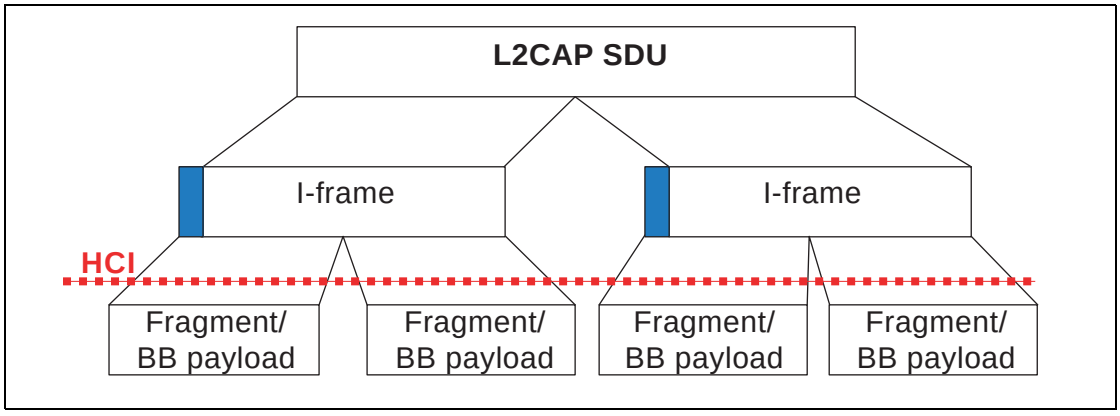



*Figure 7.3: Segmentation and fragmentation of an SDU in a BR/EDR Controller.*

### 7.3.2 Reassembly of L2CAP SDUs

The receiving side uses the SAR field of incoming 'I-frames' for the reassembly process. The L2CAP SDU length field, present in the “start of SDU” I-frame, is

an extra integrity check, and together with the sequence numbers may be used to indicate lost L2CAP SDUs to the application. Figure 7.3 on page 129 illustrates segmentation and fragmentation.

### 7.3.3 Segmentation and fragmentation

Figure 7.4 on page 131 illustrates the use of segmentation and fragmentation operations to transmit a single SDU. Note that while SDUs and L2CAP PDUs are transported in peer-to-peer fashion, the fragment size used by the Fragmentation and Recombination routines is implementation specific and may not be the same in the sender and the receiver. The over-the-air sequence of packet fragments as created by the sender is common to both devices.

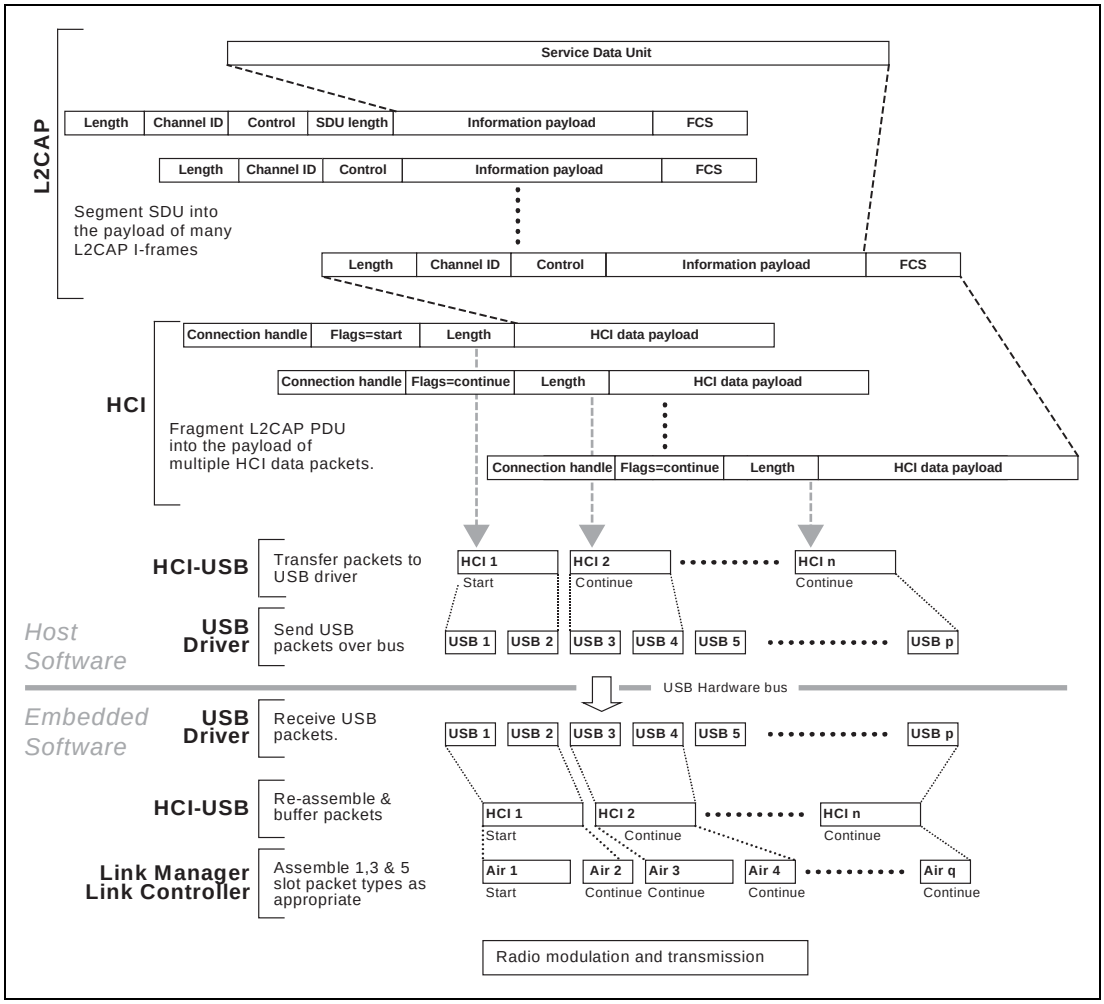



*Figure 7.4: Example of segmentation and fragment processes in a device with a BR/EDR Controller and USB HCI Transport*[4]

4. For simplicity, the stripping of any additional HCI and USB specific information fields prior to the creation of the baseband packets (Air_1, Air_2, etc.) is not shown in the figure.

## 7.4 DELIVERY OF ERRONEOUS L2CAP SDUS

Some applications may require corrupted or incomplete L2CAP SDUs to be delivered to the upper layer. If delivery of erroneous L2CAP SDUs is enabled, the receiving side will pass information to the upper layer on which parts of the L2CAP SDU (i.e., which L2CAP frames) have been lost, failed the error check, or passed the error check. If delivery of erroneous L2CAP SDUs is disabled, the receiver shall discard any L2CAP SDU segment with any missing frames or any frames failing the error checks. L2CAP SDUs whose length field does not match the actual frame length shall also be discarded.

## 7.5 OPERATION WITH FLUSHING ON ACL-U LOGICAL LINKS

In the L2CAP configuration using either the Flush Timeout option or the Extended Flow Specification option, the Flush timeout may be set separately per L2CAP channel, but in the BR/EDR baseband, the flush mechanisms operate per ACL logical transport.

When there is more than one L2CAP channel mapped to the same ACL logical transport, the automatic flush time-out does not discriminate between L2CAP channels. The automatic flush time-out also applies to unicast data sent via the L2CAP connectionless channel. The exception is packets marked as non-automatically-flushable via the Packet_Boundary_Flag in the HCI ACL Data Packet (see Section 1.1 on page 30). The automatic flush time-out flushes a specific automatically-flushable L2CAP PDU. The HCI Flush command flushes all outstanding L2CAP PDUs for the ACL logical transport including L2CAP PDUs marked as non-automatically-flushable. Therefore, care has to be taken when using the Automatic Flush Time-out and the HCI Flush command. The HCI Enhanced Flush command should be used instead.

All packets associated with a reliable connection shall be marked as non-automatically-flushable (if it is mapped to an ACL logical transport with a finite automatic flush time-out) or L2CAP Enhanced Retransmission mode or Retransmission mode shall be used. In Enhanced Retransmission mode or Retransmission mode, loss of flushed L2CAP PDUs on the channel is detected by the L2CAP ARQ mechanism and they are retransmitted. Note that L2CAP Enhanced Retransmission mode or Retransmission mode might be used for other purposes such as the need for a residual error rate that is much smaller than the baseband can deliver. In this situation L2CAP Enhanced Retransmission mode or Retransmission mode and the Non-Flushable Packet Boundary Flag feature can be used at the same time.

If it is desired to send unicast data via the L2CAP connectionless channel which is not subject to automatic flushing, then the data should be marked as non-automatically flushable if it is mapped to an ACL logical transport with a finite automatic flush time-out. Unicast data sent via the L2CAP connectionless channel may be marked flushable.

There is only one automatic flush time-out setting per ACL logical transport. Therefore, all time bounded L2CAP channels on an ACL logical transport with an automatic flush time-out setting should configure the same flush time-out value at the L2CAP level. The flush time-out setting for the ACL logical transport also applies to unicast data sent via the L2CAP connectionless channel which are marked flushable.

If Automatic Flush Time-out is used, then it should be taken into account that it only flushes one L2CAP PDU. If one PDU has timed out and needs flushing, then other automatically-flushable packets on the same logical transport are also likely to need flushing. Therefore, flushing can be handled by the HCI Enhanced Flush command so that all outstanding automatically-flushable L2CAP PDUs are flushed.

When both reliable and isochronous data is to be sent over the same ACL logical transport, an infinite Automatic Flush Time-out can be used. In this case the isochronous data is flushed using the HCI Enhanced Flush command with Packet_Type set to “Automatically-Flushable Only,” thus preserving the reliable data.

## 7.6 CONNECTIONLESS DATA CHANNEL

In addition to connection-oriented channels, L2CAP also provides a connectionless channel. Data sent on the connectionless channel shall only be sent over the BR/EDR radio. The connectionless channel allows broadcast transmissions from the master device to all members of the piconet or unicast transmissions from either a master or slave to a single remote device. Data sent through the connectionless channel is sent in a best-effort manner. The connectionless channel provided by L2CAP has no quality of service.

While L2CAP itself does not provide retransmission for data sent via the connectionless channel, the baseband does provide an ARQ scheme for unicast data (see [Vol. 2] Part B, Section 7.6). If a higher degree of reliability is desired for unicast data sent via the connectionless L2CAP channel than is provided by the baseband ARQ scheme then an ARQ scheme should be implemented at a higher layer.

No acknowledgement is provided by the baseband for broadcast transmissions and hence broadcast transmissions sent via the connectionless L2CAP channel are unreliable and hence might or might not reach each member of the piconet.

The receiving L2CAP entity may silently discard data received via the connectionless L2CAP channel if the data packet is addressed to a PSM and no application is registered to receive data on that PSM.

Note that L2CAP does not provide flow control for either connection-oriented L2CAP channels operating in Basic mode or for traffic on the connectionless L2CAP channel. Hence if data received by L2CAP is not accepted by the target

applications in a timely manner, congestion can occur in a receiving L2CAP implementation. For connection-oriented channels, the receiving L2CAP entity may elect to close a channel if the target application for that channel does not accept the data in a timely manner. Since this option is not available for unicast data received via the connectionless L2CAP channel, the receiving L2CAP entity may instead elect to de-register the application receiving the data or to disconnect the underlying physical link. An application shall be notified if it is de-registered. If the underlying physical link is disconnected then all applications utilizing that physical link shall be notified.

Unicast data shall only be sent via the connectionless L2CAP channel to a remote device if the remote device indicates support for Unicast Connectionless Data Reception in the L2CAP Extended Features Mask.[5]

Unicast data sent via the L2CAP connectionless channel are subject to automatic flushing and hence the packet boundary flags should be set appropriately if the controller supports the Packet Boundary Flag feature. See Section 7.5 for further details. Since the receiving L2CAP entity has no mechanism to enable it to know whether received packets were originally marked as flushable or non-flushable on the transmitting device, the receiving L2CAP entity should treat all unicast packets received via the connectionless L2CAP packet as non-flushable.

If encryption is required for a given profile, then the profile or application shall ensure that authentication is performed and encryption is enabled prior to sending any unicast data on the connectionless L2CAP channel by utilizing Security Mode 4 as defined in GAP ([Vol. 3] Part C, Section 5.2.2). There is no mechanism provided in this specification to prevent reception of unencrypted data on the connectionless L2CAP channel. An application which requires received data to be encrypted should ignore any unencrypted data it may receive over the connectionless channel.

Broadcast transmissions to the connectionless channel are sent with the broadcast LT_ADDR and hence may be received by any of the slaves in the piconet. If it is desirable to restrict the reception of the transmitted data to only a subset of the slaves in the piconet, then higher level encryption may be used to support private communication.

The master will not receive transmissions broadcast on the connectionless channel. Therefore, higher layer protocols must loopback any broadcast data traffic being sent to the master device if required.

5. Note that the Unicast Connectionless Data Reception bit in the L2CAP Extended Features Mask does not in any way indicate support or lack of support for reception of broadcast data on the connectionless L2CAP channel even though both broadcast data and unicast data are sent and received using the same CID (0x0002). For historical reasons, there is no bit to indicate support for sending or receiving of broadcast data on the connectionless L2CAP channel.

The connectionless data channel shall not be used with Enhanced Retransmission mode, Retransmission Mode or Flow Control Mode.

## 7.7 OPERATION COLLISION RESOLUTION

When two devices request the same operation by sending a request packet with the same code, a collision can occur. Some operations require collision resolution. The description of each operation in Section 4 will indicate if collision resolution is required. Both devices must know which request will be rejected. The following algorithm shall be used by both devices to determine which request to reject.

1. Set i=0 (representing the least significant octet of the BD_ADDR).
2. Compare the octet[i] of the BD_ADDR of both devices. If the octets are not equal go to step 4.
3. Increment i by 1. Go to step 2.
4. The device with the larger BD_ADDR octet shall reject the request from the other device.

## 7.8 AGGREGATING BEST EFFORT EXTENDED FLOW SPECIFICATIONS

There shall only be one pair of Extended Flow Specifications per logical link and only one “Best Effort” logical link per physical link. Thus, all Best Effort L2CAP channels running over the same physical link are aggregated into one logical link and the Best Effort Flow Specifications describing the traffic for each channel running over an AMP-U logical link shall be aggregated to a single pair of Extended Flow Specifications (one for each direction). The purpose of a Best Effort Flow Specification is to specify the maximum data rate of the data that can be delivered to the Controller from the host and/or taken from the Controller by the host in order to help the Controller to not over allocate bandwidth on the physical link. Since L2CAP performs admission control for BR/EDR and BR/EDR/LE Controllers it is not necessary to aggregate Best Effort channels for BR/EDR and BR/EDR/LE Controllers.

The two Extended Flow Specification parameters that are affected by Best Effort are Maximum SDU size and SDU Inter-arrival Time. A data rate is determined by dividing the Maximum SDU size by the SDU Inter-arrival time giving a value in bytes per second. The important value is the data rate and not the Maximum SDU size or SDU Inter-arrival time.

Best Effort aggregation shall occur when performing any of the three operations:

- Configuring a new Best Effort channel
- Disconnecting a Best Effort channel
- Moving a Best Effort a channel

During L2CAP channel configuration, the Extended Flow Specifications are exchanged and each L2CAP entity is allowed to modify the other's flow specification (lower the values). If a Best Effort logical link already exists (meaning there is at least one other Best Effort channel) then the new flow specifications shall be aggregated with the existing Best Effort aggregate before sending it to the Controller. When a Best Effort channel is moved or disconnected a new aggregate shall be created. If the new aggregate is different from the previous aggregate it shall be given to the Controller. The following guidelines are provided for aggregating Best Effort Flow specification parameters. Keep in mind that the data rate is the important value so the description shows how to aggregate data rates.

1. The value 0xFFFF is used for Maximum SDU Size and the value 0xFFFFFFFF is used for SDU Inter-arrival time to indicate that the actual values are unknown and maximum bandwidth is assumed. Therefore, if these values exist in an Extended Flow Specification then the aggregate becomes 0xFFFF and 0xFFFFFFFF for Maximum SDU Size and SDU Inter-arrival time respectively.
2. In general it is assumed that data rates provided by the local applications can be added together. For example if one application indicates a data rate of 100 bytes/sec and a second application indicates a data rate of 100 bytes/sec then the aggregate data rate from the two applications is 200 bytes/sec. This is assumed for both directions.
3. The adjustments (sent in a Configuration Response) made by the remote device to an outgoing flow specification are assumed to be made on behalf of the application. These adjustments reduce the data rate of the outgoing flow specification. The reduced data rates can be added to data rates from other applications.
4. The final data rates shall be converted back into Maximum SDU Size and SDU Inter-arrival time to be passed to the Controller. Controllers may have a maximum PDU size or a “preferred” PDU size. When possible the “preferred” size should be used for the Maximum SDU size and the SDU Inter-arrival time should be set to a value that achieves the desired data rate. If the data rate cannot be achieved by using the “preferred” PDU size then an integer multiple of the preferred PDU size should be used.

## 7.9 PRIORITIZING DATA OVER HCI

In order for guaranteed channels to meet their guarantees, L2CAP should prioritize traffic over the HCI transport in devices that support HCI. Packets for Guaranteed channels should receive higher priority than packets for Best Effort channels.

## 7.10 SUPPORTING EXTENDED FLOW SPECIFICATION FOR BR/EDR AND BR/EDR/LE CONTROLLERS

If both the local L2CAP entity and the remote L2CAP entity indicate support for Extended Flow Specification for BR/EDR in the Extended Feature Mask then all channels created between the two devices shall send an Extended Flow Specification option and shall use the Lockstep configuration procedure. In addition all channels shall use Enhanced Retransmission mode or Streaming mode. If one or both L2CAP entities do not indicate support for Extended Flow Specification for BR/EDR in the Extended Feature Mask then the Lockstep configuration procedure shall not be used and the Extended Flow Specification option shall not be sent for channels created over the ACL-U logical link between the two devices.

The L2CAP entity shall perform admission control for Guaranteed channels during the Lockstep configuration procedure (see Section 7.1.3). Admission control is performed to determine if the requested Guaranteed QoS can be achieved by the BR/EDR or BR/EDR/LE Controller over an ACL-U logical link without compromising existing Guaranteed channels running on the Controller. If HCI is used then the L2CAP entity shall also verify that the QoS can be achieved over the HCI transport.

In performing admission control the L2CAP layer shall reject a Guaranteed QoS request that causes at least one of the following rules to be violated.

1. The total guaranteed data rate in both directions shall not exceed two times the highest Symmetric Max of an ACL-U logical link over the BR/EDR or BR/EDR/LE Controller (see [vol.2, part B] Section 6.7 on page 132). For example for a Basic Rate Controller the highest Symmetric Max Rate is 433.9kb/s (DH5) from Table 6.9 in [vol.2, part B] Section 6.7 on page 132. Two times that value is 867.8kb/s.
2. The total guaranteed data rate in any one direction shall not exceed the highest Asymmetric Max Rate of an ACL-U logical link (see [Vol. 2] Part B, Section 6.7). For example the highest Asymmetric Max Rate for Basic Rate is 723.2kb/s (DH5 packet) from Table 6.9 in [vol.2, part B] Section 6.7 on page 132.

The data rate of a Guaranteed channel is calculated by dividing the Maximum SDU size in an Extended Flow Specification by the SDU Inter-arrival time. Total guaranteed data rate in both directions is calculated by taking the sum of the data rates in both directions for all existing Guaranteed channels plus the data rate in both directions of the requested Guaranteed channel (i.e. data rates from the both outgoing and incoming Extended Flow Specifications). Total guaranteed data rate for one direction is calculated by taking the sum of the data rates in one direction for all existing Guaranteed channels plus the data rate in the same direction of the requested Guaranteed channel.

An L2CAP entity should use additional information for admission control that may result in a Guaranteed QoS request being rejected even if none of the

rules are violated. This allows the L2CAP entity to account for things such as CQDDR, SCO/eSCO channels, LMP traffic, page scanning, etc.

In order to meet the access latency and/or data rate required by a Guaranteed channel, the L2CAP entity should:

a) Prioritize traffic over the HCI transport in devices that support HCI.

b) Give conformant packets for Guaranteed channels higher priority than packets for Best Effort channels. Packets for Best Effort channels should have higher priority than non-conformant packets for Guaranteed channels. (See Section 5.6 for a discussion of non-conformant and conformant traffic).

c) Utilize the SAR feature to restrict the PDU sizes of Best Effort SDUs so that they do not prevent the Controller from sending the SDUs of the Guaranteed channel within the time periods required by its Extended Flow Specification.

When a channel is reconfigured the Lockstep configuration process is only used when an Extended Flow Specification option is present in the Configuration Request packet as described in Section 7.1.3.

# 8 PROCEDURES FOR FLOW CONTROL AND RETRANSMISSION

When Enhanced Retransmission mode, Streaming mode, Flow Control mode, or Retransmission mode is used, the procedures defined in this chapter shall be used, including the numbering of information frames, the handling of SDU segmentation and reassembly, and the detection and notification of errored frames. Retransmission mode and Enhanced Retransmission mode also allow the sender to resend errored frames on request from the receiver.

## 8.1 INFORMATION RETRIEVAL

Before attempting to configure Enhanced Retransmission mode, Streaming mode, Flow Control mode, or Retransmission mode on a channel, support for the suggested mode shall be verified by performing an information retrieval for the “Extended features supported” information type (0x0002). If the information retrieval is not successful or the “Extended features mask” bit is not set for the wanted mode, the mode shall not be suggested in a configuration request.

## 8.2 FUNCTION OF PDU TYPES FOR FLOW CONTROL AND RETRANSMISSION

Two frame formats are defined for Enhanced Retransmission mode, Streaming Mode, Flow Control Mode, and Retransmission mode (see Section 3.3 on page 45). The I-frame is used to transport user information instead of the B-frame. The S-frame is used for supervision.

### 8.2.1 Information frame (I-frame)

I-frames are sequentially numbered frames containing information fields. I-frames also include some of the functionality of RR frames (see below).

### 8.2.2 Supervisory Frame (S-frame)

The S-frame is used to control the transmission of I-frames. For Retransmission mode and Flow Control mode, the S-frame has two formats: Receiver Ready (RR) and Reject (REJ). A description of how S-frames are used in Enhanced Retransmission mode is given in Section 8.6.1. S-frames are not used in Streaming mode. The following description of S-frames only applies to Retransmission mode and Flow Control mode.

#### *8.2.2.1 Receiver Ready (RR)*

The receiver ready (RR) S-frame is used to:

1. Acknowledge I-frames numbered up to and including ReqSeq - 1.
2. Enable or disable retransmission of I-frames by updating the receiver with the current status of the Retransmission Disable Bit.

The RR frame has no information field.

#### *8.2.2.2 Reject (REJ)*

The reject (REJ) S-frame is used to request retransmission of all I-frames starting with the I-frame with TxSeq equal to ReqSeq specified in the REJ. The value of ReqSeq in the REJ frame acknowledges I-frames numbered up to and including ReqSeq - 1. I-frames that have not been transmitted, shall be transmitted following the retransmitted I-frames.

When a REJ is transmitted, it triggers a REJ Exception condition. A second REJ frame shall not be transmitted until the REJ Exception condition is cleared. The receipt of an I-frame with a TxSeq equal to the ReqSeq of the REJ frame clears the REJ Exception. The REJ Exception condition only applies to traffic in one direction. Note: this means that only valid I-frames can be rejected.

## 8.3 VARIABLES AND SEQUENCE NUMBERS

The sending peer uses the following variables and Sequence numbers:

- TxSeq – the send Sequence number used to sequentially number each new I-frame transmitted.
- NextTxSeq – the Sequence number to be used in the next new I-frame transmitted.
- ExpectedAckSeq – the Sequence number of the next I-frame expected to be acknowledged by the receiving peer.

The receiving peer uses the following variables and Sequence numbers:

- ReqSeq – The Sequence number sent in an acknowledgement frame to request transmission of I-frame with TxSeq = ReqSeq and acknowledge receipt of I-frames up to and including (ReqSeq-1)
- ExpectedTxSeq – the value of TxSeq expected in the next I-frame.
- BufferSeq – When segmented I-frames are buffered this is used to delay acknowledgement of received I-frame so that new I-frame transmissions do not cause buffer overflow.

All variables have the range 0 to 63. Arithmetic operations on state variables (NextTXSeq, ExpectedTxSeq, ExpectedAckSeq, BufferSeq) and sequence numbers (TxSeq, ReqSeq) contained in this document shall be taken modulo 64.

To perform Modulo 64 operation on negative numbers multiples of 64 shall be added to the negative number until the result becomes non-negative.

### 8.3.1 Sending peer

#### *8.3.1.1 Send sequence number TxSeq*

I-frames contain TxSeq, the send sequence number of the I-frame. When an I-frame is first transmitted, TxSeq is set to the value of the send state variable NextTXSeq. TxSeq is not changed if the I-frame is retransmitted.

#### *8.3.1.2 Send state variable NextTXSeq*

The CID sent in the information frame is the destination CID and identifies the remote endpoint of the channel. A send state variable NextTxSeq shall be maintained for each remote endpoint. NextTxSeq is the sequence number of the next in-sequence I-frame to be transmitted to that remote endpoint. When the link is created NextTXSeq shall be initialized to 0.

The value of NextTxSeq shall be incremented by 1 after each in-sequence I-frame transmission, and shall not exceed ExpectedAckSeq by more than the maximum number of outstanding I-frames (TxWindow). The value of TxWindow shall be in the range 1 to 32 for Retransmission mode and Flow Control mode. The value of TxWindow shall be in the range of 1 to 63 for Enhanced Retransmission mode.

#### *8.3.1.3 Acknowledge state variable ExpectedAckSeq*

The CID sent in the information frame is the destination CID and identifies the remote endpoint of the channel. An acknowledge state variable ExpectedAckSeq shall be maintained for each remote endpoint. ExpectedAckSeq is the sequence number of the next in-sequence I-frame that the remote receiving peer is expected to acknowledge. (ExpectedAckSeq – 1 equals the TxSeq of the last acknowledged I-frame). When the link is created ExpectedAckSeq shall be initialized to 0.
Note that if the next acknowledgement acknowledges a single I-frame then its ReqSeq will be expectedAckSeq + 1.

If a valid ReqSeq is received from the peer then ExpectedAckSeq is set to ReqSeq. A valid ReqSeq value is one that is in the range ExpectedAckSeq ≤ ReqSeq ≤ NextTxSeq.

Note: The comparison with NextTXSeq must be ≤ in order to handle the situations where there are no outstanding I-frames.

These inequalities shall be interpreted in the following way: ReqSeq is valid, if and only if (ReqSeq-ExpectedAckSeq) mod 64 ≤ (NextTXSeq-ExpectedAckSeq) mod 64. Furthermore, from the description of NextTXSeq, it can be seen that (NextTXSeq-ExpectedAckSeq) mod 64 ≤ TxWindow.

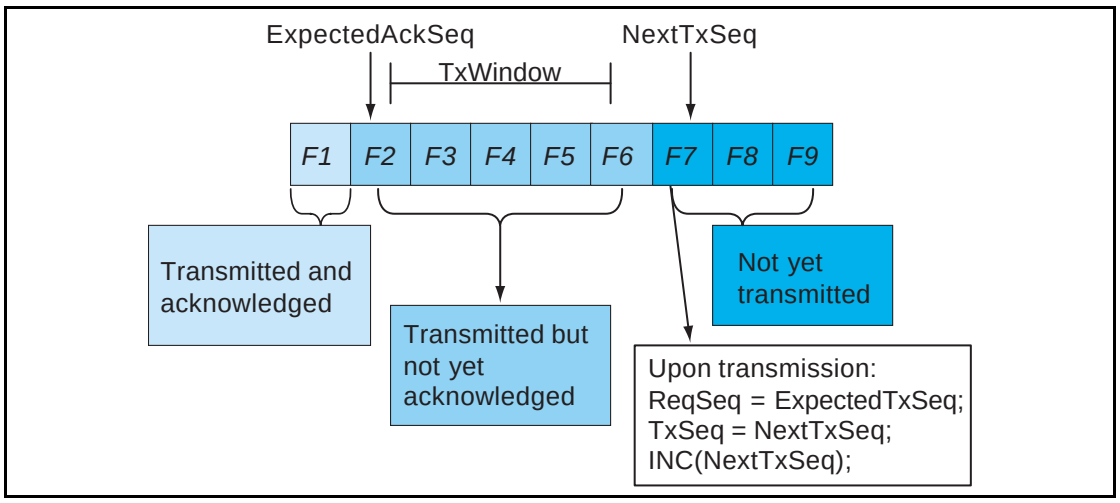



*Figure 8.1: Example of the transmitter side*

Figure 8.1 on page 142 shows TxWindow=5, and three frames awaiting transmission. The frame with number *F7* may be transmitted when the frame with *F2* is acknowledged. When the frame with *F7* is transmitted, TxSeq is set to the value of NextTXSeq. After TxSeq has been set, NextTxSeq is incremented.

The sending peer expects to receive legal ReqSeq values, these are in the range ExpectedAckSeq up to and including NextTxSeq. Upon receipt of a ReqSeq value equal to the current NextTxSeq all outstanding I-frames have been acknowledged by the receiver.

### 8.3.2 Receiving peer

#### *8.3.2.1 Receive sequence number ReqSeq*

All I-frames and S-frames contain ReqSeq, the send Sequence number (TxSeq) that the receiving peer requests in the next I-frame.

When an I-frame or an S-frame is transmitted, the value of ReqSeq shall be set to the current value of the receive state variable ExpectedTxSeq or the buffer state variable BufferSeq. The value of ReqSeq shall indicate that the data link layer entity transmitting the ReqSeq has correctly received all I-frames numbered up to and including ReqSeq – 1.

Note: The option to set ReqSeq to BufferSeq instead of ExpectedTxSeq allows the receiver to impose flow control for buffer management or other purposes. In this situation, if BufferSeq<>ExpectedTxSeq, the receiver should also set the retransmission disable bit to 1 to prevent unnecessary retransmissions.

### *8.3.2.2 Receive state variable, ExpectedTxSeq*

Each channel shall have a receive state variable (ExpectedTxSeq). The receive state variable is the sequence number (TxSeq) of the next in-sequence I-frame expected.

The value of the receive state variable shall be the last in-sequence, valid I-frame received.

### *8.3.2.3 Buffer state variable BufferSeq*

Each channel may have an associated BufferSeq. BufferSeq is used to delay acknowledgement of frames until they have been pulled by the upper layers, thus preventing buffer overflow. BufferSeq and ExpectedTxSeq are equal when there is no extra segmentation performed and frames are pushed to the upper layer immediately on reception. When buffer space is scarce, for example when frames reside in the buffer for a period, the receiver may choose to set ReqSeq to BufferSeq instead of ExpectedTxSeq, incrementing BufferSeq as buffer space is released. The windowing mechanism will ensure that transmission is halted when ExpectedTxSeq - BufferSeq is equal to TxWindow.

Note: Owing to the variable size of I-frames, updates of BufferSeq may be based on changes in available buffer space instead of delivery of I-frame contents.

I-frames shall have sequence numbers in the range ExpectedTxSeq ≤ TxSeq < (BufferSeq + TxWindow).

On receipt of an I-frame with TxSeq equal to ExpectedTxSeq, ExpectedTxSeq shall be incremented by 1 regardless of how many I-frames with TxSeq greater than ExpectedTxSeq were previously received.

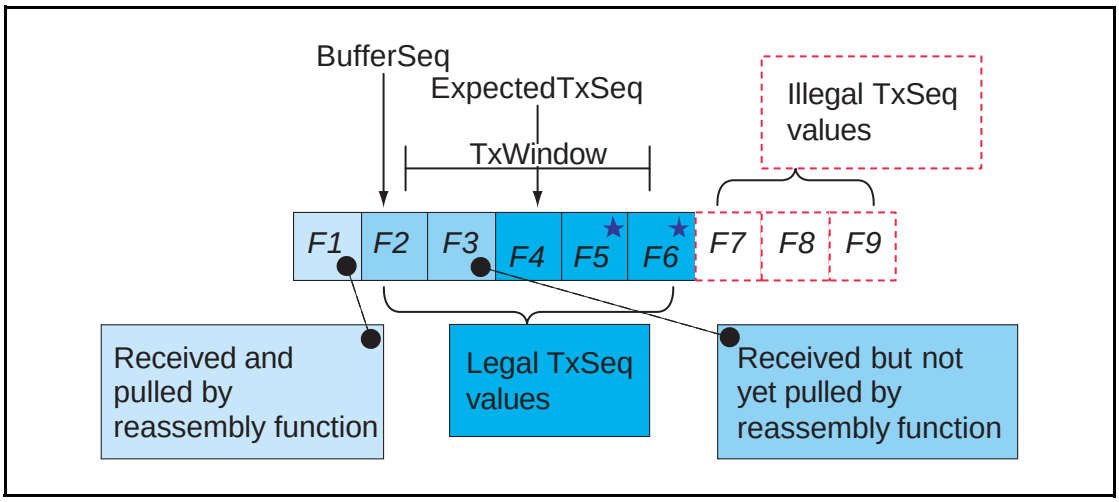



*Figure 8.2: Example of the receiver side*

Figure 8.2 on page 143 shows TxWindow=5. *F1* is successfully received and pulled by the upper layer. BufferSeq shows that *F2* is the next I-frame to be pulled, and ExpectedTxSeq points to the next I-frame expected from the peer. An I-frame with TxSeq equal to 5 has been received thus triggering an SREJ or REJ exception. The star indicates I-frames received but discarded owing to the SREJ or REJ exception. They will be resent as part of the error recovery procedure.

In Figure 8.2 there are several I-frames in a buffer awaiting the SDU reassembly function to pull them and the TxWindow is full. The receiver would usually disable retransmission in Retransmission mode or Flow Control mode by setting the Retransmission Disable Bit to 1 and send an RR back to the sending side. This tells the transmitting peer that there is no point in performing retransmissions. Both sides will send S-frames to make sure the peer entity knows the current value of the Retransmission Disable Bit.

## 8.4 RETRANSMISSION MODE

### 8.4.1 Transmitting frames

A new I-frame shall only be transmitted when the TxWindow is not full. No I-frames shall be transmitted if the last RetransmissionDisableBit (R) received is set to one.

A previously transmitted I-frame may be retransmitted as a result of an error recovery procedure, even if the TxWindow is full. When an I-frame is retransmitted it shall always be sent with the same TxSeq value used in its initial transmission.

The state of the RetransmissionDisableBit (R) is stored and used along with the state of the RetransmissionTimer to decide the actions when transmitting I-frames. The RetransmissionTimer is running whenever I-frames have been sent but not acknowledged.

#### *8.4.1.1 Last received R was set to zero*

If the last R received was set to zero, then I-frames may be transmitted. If there are any I-frames which have been sent and not acknowledged then they shall be retransmitted when the RetransmissionTimer elapses. If the retransmission timer has not elapsed then a retransmission shall not be sent and only new I-frames may be sent.

a) If unacknowledged I-frames have been sent and the RetransmissionTimer has elapsed then an unacknowledged I-

frame shall be retransmitted. The RetransmissionTimer shall be restarted.

b) If unacknowledged I-frames have been sent, the Retransmission timer has not elapsed then a new I-frame shall be sent if one is waiting and no timer action shall be taken.

c) If no unacknowledged I-frames have been sent, and a new I-frame is waiting, then the new I-frame shall be sent, the RetransmissionTimer shall be started and if the Monitor Timer is running, it shall be stopped.

d) If no unacknowledged I-frames have been sent and no new I-frames are waiting to be transmitted, and the RetransmissionTimer is running, then the retransmission timer shall be stopped and the monitor timer shall be started.

The table below summarizes actions when the RetransmissionTimer is in use and R=0.

| **Unacknowledged I-frames sent = Retransmission Timer is running** | **Retrans-mission Timer has elapsed** | **New I-frames are waiting** | **Transmit Action** | **Timer Action** |
|---|---|---|---|---|
| True | True | True or False | Retransmit un-acknowledged I-frame | Restart Retransmission Timer |
| True | False | True | Transmit new I-frame | No timer action |
| True | False | False | No transmit action | No Timer action |
| False | False | True | Transmit new I-frame | Restart Retransmis-sion Timer |
| False | False | False | No Transmit action | If MonitorTimer is not running then restart MonitorTimer |

*Table 8.1: Summary of actions when the RetransmissionTimer is in use and R=0.*

If the RetransmissionTimer is not in use, no unacknowledged I-frames have been sent and no new I-frames are waiting to be transmitted

a) If the MonitorTimer is running and has not elapsed then no transmit action shall be taken and no timer action shall be taken.

b) If the MonitorTimer has elapsed then an S-frame shall be sent and the MonitorTimer shall be restarted.

If any I-frames become available for transmission then the MonitorTimer shall be stopped, the RetransmissionTimer shall be started and the rules for when the RetransmissionTimer is in use shall be applied.

When an I-frame is sent ReqSeq shall be set to ExpectedTxSeq, TxSeq shall be set to NextTxSeq and NextTxSeq shall be incremented by one.

#### *8.4.1.2 Last received R was set to one*

If the last R received was set to one, then I-frames shall not be transmitted. The only frames which may be sent are S-frames. An S-frame shall be sent according to the rules below:

a) If the MonitorTimer is running and has not elapsed then no transmit action shall be taken and no timer action shall be taken.

b) If the MonitorTimer has elapsed then an S-frame shall be sent and the MonitorTimer shall be restarted.

### 8.4.2 Receiving I-frames

Upon receipt of a valid I-frame with TxSeq equal to ExpectedTxSeq, the frame shall be accepted for the SDU reassembly function. ExpectedTxSeq is used by the reassembly function.

The first valid I-frame received after an REJ was sent, with a TxSeq of the received I-frame equal to ReqSeq of the REJ, shall clear the REJ Exception condition.

The ReqSeq shall be processed according to Section 8.4.6 on page 149.

If a valid I-frame with TxSeq ≠ ExpectedTxSeq is received then an exception condition shall be triggered which is handled according to Section 8.4.7 on page 149.

### 8.4.3 I-frames pulled by the SDU reassembly function

When the L2CAP layer has removed one or more I-frames from the buffer, BufferSeq may be incremented in accordance with the amount of buffer space released. If BufferSeq is incremented, an acknowledgement shall be sent to the peer entity.

Note: Since the primary purpose of BufferSeq is to prevent buffer overflow, an implementation may choose to set BufferSeq in accordance with how many new incoming I-frames could be stored rather than how many have been removed.

The acknowledgement may either be an RR or an I-frame. The acknowledgement shall be sent to the peer L2CAP entity with ReqSeq equal to BufferSeq. When there are no I-frames buffered for pulling ExpectedTxSeq is equal to BufferSeq.

If the MonitorTimer is active then it shall be restarted to indicate that a signal has been sent to the peer L2CAP entity.

### 8.4.4 Sending and receiving acknowledgements

Either the MonitorTimer or the RetransmissionTimer shall be active while in Retransmission Mode. Both timers shall not be active concurrently.

#### *8.4.4.1 Sending acknowledgements*

Whenever an L2CAP entity transmits an I-frame or an S-frame, ReqSeq shall be set to ExpectedTxSeq or BufferSeq.

#### *8.4.4.2 Receiving acknowledgements*

On receipt of a valid S-frame or I-frame, the ReqSeq contained in the frame shall acknowledge previously transmitted I-frames. ReqSeq acknowledges I-frames with a TxSeq up to and including ReqSeq – 1.

The following rules shall be applied:

1. If the RetransmissionDisableBit changed value from 0 to 1 (stop retransmissions) then the receiving entity shall
    a) If the RetransmissionTimer is running then stop it and start the MonitorTimer.
    b) Store the state of the RetransmissionDisableBit received.
2. If the RetransmissionDisableBit changed value from 1 to 0 (start retransmissions) then the receiving entity shall
    a) Store the state of the RetransmissionDisableBit received.
    b) If there are any I-frames that have been sent but not acknowledged, then stop the MonitorTimer and start the RetransmissionTimer.
    c) Any buffered I-frames shall be transmitted according to Section 8.4.1 on page 144.
3. If any unacknowledged I-frames were acknowledged by the ReqSeq contained in the frame, and the RetransmissionDisableBit equals 1 (retransmissions stopped), then the receiving entity shall
    a) Follow the rules in Section 8.4.1 on page 144.
4. If any unacknowledged I-frames were acknowledged by the ReqSeq contained in the frame and the RetransmissionDisableBit equals 0 (retransmissions started) then the receiving entity shall
    a) If the RetransmissionTimer is running, then stop it.
    b) If any unacknowledged I-frames have been sent then the RetransmissionTimer shall be restarted.
    c) Follow the rules in Section 8.4.1 on page 144.
    d) If the RetransmissionTimer is not running and the MonitorTimer is not running, then start the MonitorTimer.

On receipt of a valid S-frame or I-frame the ReqSeq contained in the frame shall acknowledge previously transmitted I-frames. ExpectedAckSeq shall be set to ReqSeq to indicate that the I-frames with TxSeq up to and including (ReqSeq - 1) have been acknowledged.

### 8.4.5 Receiving REJ frames

Upon receipt of a valid REJ frame, where ReqSeq identifies an I-frame not yet acknowledged, the ReqSeq acknowledges I-frames with TxSeq up to and including ReqSeq - 1. Therefore the REJ acknowledges all I-frames before the I-frame it is rejecting.

ExpectedAckSeq shall be set equal to ReqSeq to mark I-frames up to and including ReqSeq - 1 as received.

NextTXSeq shall be set to ReqSeq to cause transmissions of I-frames to resume from the point where TxSeq equals ReqSeq.

If ReqSeq equals ExpectedAckSeq then the REJ frame shall be ignored.

### 8.4.6 Waiting acknowledgements

A counter, TransmitCounter, counts the number of times an L2CAP PDU has been transmitted. This shall be set to 1 after the first transmission. If the RetransmissionTimer expires the following actions shall be taken:

1. If the TransmitCounter is less than MaxTransmit then:
    a) Increment the TransmitCounter
    b) Retransmit the last unacknowledged I-frame, according to Section 8.4.1 on page 144.
2. If the TransmitCounter is equal to MaxTransmit this channel to the peer entity shall be assumed lost. The channel shall move to the CLOSED state and appropriate action shall be taken to report this to the upper layers.

### 8.4.7 Exception conditions

Exception conditions may occur as the result of physical layer errors or L2CAP procedural errors. The error recovery procedures which are available following the detection of an exception condition at the L2CAP layer in Retransmission Mode are defined in this section.

#### *8.4.7.1 TxSeq Sequence error*

A TxSeq sequence error exception condition occurs in the receiver when a valid I-frame is received which contains a TxSeq value which is not equal to the expected value, thus TxSeq is not equal to ExpectedTxSeq.

The TxSeq sequence error may be due to three different causes:

- *Duplicated I-frame*

  The duplicated I-frame is identified by a TxSeq in the range BufferSeq to ExpectedTxSeq – 1 (BufferSeq ≤TxSeq<ExpectedTxSeq). The ReqSeq and RetransmissionDisableBit shall be processed according to Section 8.4.4 on page 147. The Information field shall be discarded since it has already been received.

- *Out-of-sequence I-frame*

  The out-of-sequence I-frame is identified by a TxSeq within the legal range. The ReqSeq and RetransmissionDisableBit shall be processed according to Section 8.4.4 on page 147.

  A REJ exception is triggered, and an REJ frame with ReqSeq equal to ExpectedTxSeq shall be sent to initiate recovery. The received I-frame shall be discarded.

- *Invalid TxSeq*

An invalid TxSeq value is a value that does not meet either of the above conditions. An I-frame with an invalid TxSeq is likely to have errors in the control field and shall be silently discarded.

#### *8.4.7.2 ReqSeq Sequence error*

An ReqSeq sequence error exception condition occurs in the transmitter when a valid S-frame or I-frame is received which contains an invalid ReqSeq value. An invalid ReqSeq is one that is not in the range ExpectedAckSeq ≤ ReqSeq ≤ NextTxSeq.

The L2CAP entity shall close the channel as a consequence of an ReqSeq Sequence error.

#### *8.4.7.3 Timer recovery error*

If an L2CAP entity fails to receive an acknowledgement for the last I-frame sent, then it will not detect an out-of-sequence exception condition and therefore will not transmit an REJ frame.

The L2CAP entity that transmitted an unacknowledged I-frame shall, on the expiry of the RetransmissionTimer, take appropriate recovery action as defined in Section 8.4.6 on page 149.

#### *8.4.7.4 Invalid frame*

Any frame received which is invalid (as defined in Section 3.3.6 on page 52) shall be discarded, and no action shall be taken as a result of that frame.

## 8.5 FLOW CONTROL MODE

When a link is configured to work in flow control mode, the flow control operation is similar to the procedures in retransmission mode, but all operations dealing with CRC errors in received packets are not used. Therefore

- REJ frames shall not be used in Flow Control Mode.
- The RetransmissionDisableBit shall always be set to zero in the transmitter, and shall be ignored in the receiver.

The behavior of flow control mode is specified in this section.

Assuming that the TxWindow size is equal to the buffer space available in the receiver (counted in number of I-frames), in flow control mode the number of unacknowledged frames in the transmitter window is always less than or equal to the number of frames for which space is available in the receiver. Note that a missing frame still occupies a place in the window.

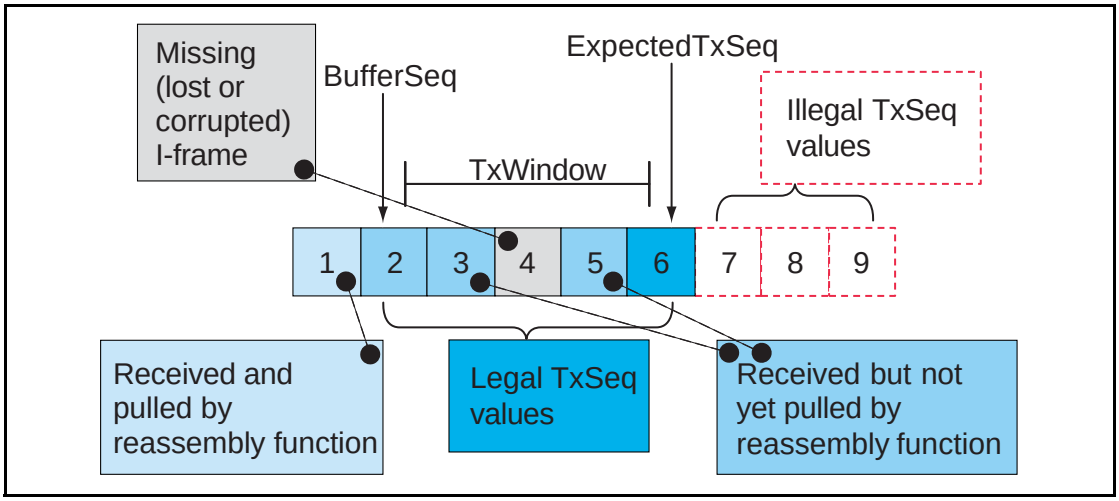



*Figure 8.3: Overview of the receiver side when operating in flow control mode*

### 8.5.1 Transmitting I-frames

A new I-frame shall only be transmitted when the TxWindow is not full.

Upon transmission of the I-frame the following actions shall be performed:

- If no unacknowledged I-frames have been sent then the MonitorTimer shall be stopped and the RetransmissionTimer shall be started.
- If any I-frames have been sent and not acknowledged then the RetransmissionTimer remains active and no timer operation is performed.

The control field parameter ReqSeq shall be set to ExpectedTxSeq, TxSeq shall be set to NextTXSeq and NextTXSeq shall be incremented by one.

### 8.5.2 Receiving I-frames

Upon receipt of a valid I-frame with TxSeq equal to ExpectedTxSeq, the frame shall be made available to the reassembly function. ExpectedTxSeq shall be incremented by one. An acknowledgement shall not be sent until the SDU reassembly function has pulled the I-frame.

Upon receipt of a valid I-frame with an out-of-sequence TxSeq (see Section 8.5.6 on page 153) all frames with a sequence number less than TxSeq shall be assumed lost and marked as missing. The missing I-frames are in the range from ExpectedTxSeq (the frame that the device was expecting to receive) up to TxSeq-1, (the frame that the device actually received). ExpectedTxSeq shall be set to TxSeq +1. The received I-frame shall be made available for pulling by the reassembly function. The acknowledgement shall not occur until the SDU reassembly function has pulled the I-frame. The ReqSeq shall be processed according to Section 8.5.4 on page 152.

### 8.5.3 I-frames pulled by the SDU reassembly function

When the L2CAP layer has removed one or more I-frames from the buffer, BufferSeq may be incremented in accordance with the amount of buffer space released. If BufferSeq is incremented, an acknowledgement shall be sent to the peer entity. If the MonitorTimer is active then it shall be restarted to indicate that a signal has been sent to the peer L2CAP entity.

Note: Since the primary purpose of BufferSeq is to prevent buffer overflow, an implementation may choose to set BufferSeq in accordance with how many new incoming I-frames could be stored rather than how many have been removed.

The acknowledgement may be an RR or an I-frame. The acknowledgement shall be sent to the peer L2CAP entity with ReqSeq equal to BufferSeq. When there is no I-frame buffered for pulling, ExpectedTxSeq is equal to BufferSeq.

### 8.5.4 Sending and receiving acknowledgements

One of the timers MonitorTimer or RetransmissionTimer shall always be active while in Flow Control mode. Both timers shall never be active concurrently.

#### *8.5.4.1 Sending acknowledgements*

Whenever a data link layer entity transmits an I-frame or a S-frame, ReqSeq shall be set to ExpectedTxSeq or BufferSeq.

#### *8.5.4.2 Receiving acknowledgements*

On receipt of a valid S-frame or I-frame the ReqSeq contained in the frame shall be used to acknowledge previously transmitted I-frames. ReqSeq acknowledges I-frames with a TxSeq up to and including ReqSeq – 1.

1. If any outstanding I-frames were acknowledged then
    a) Stop the RetransmissionTimer
    b) If there are still unacknowledged I-frames then restart the RetransmissionTimer, otherwise start the MonitorTimer.
    c) Transmit any I-frames awaiting transmission according to Section 8.5.1 on page 151.

ExpectedAckSeq shall be set to ReqSeq to indicate that the I-frames with TxSeq up to and including ExpectedAckSeq have been acknowledged.

### 8.5.5 Waiting acknowledgements

If the RetransmissionTimer expires the following actions shall be taken:
The I-frame supervised by the RetransmissionTimer shall be considered lost, and ExpectedAckSeq shall be incremented by one.

1. If I-frames are waiting to be sent
    a) The RetransmissionTimer is restarted
    b) I-frames awaiting transmission are transmitted according to Section 8.5.1 on page 151.
2. If there are no I-frames waiting to be sent
    a) If there are still unacknowledged I-frames the RetransmissionTimer is restarted, otherwise the MonitorTimer is started.

### 8.5.6 Exception conditions

Exception conditions may occur as the result of physical layer errors or L2CAP procedural errors. The error recovery procedures which are available following the detection of an exception condition at the L2CAP layer in flow control only mode are defined in this section.

#### *8.5.6.1 TxSeq Sequence error*

A TxSeq sequence error exception condition occurs in the receiver when a valid I-frame is received which contains a TxSeq value which is not equal to the expected value, thus TxSeq is not equal to ExpectedTxSeq.

The TxSeq sequence error may be due to three different causes:

- *Duplicated I-frame*

  The duplicated I-frame is identified by a TxSeq in the range BufferSeq to ExpectedTxSeq – 1. The ReqSeq shall be processed according to Section 8.5.4 on page 152. The Information field shall be discarded since it has already been received.

- *Out-of-sequence I-frame*

  The out-of-sequence I-frame is identified by a TxSeq within the legal range ExpectedTxSeq < TxSeq < (BufferSeq + TxWindow). The ReqSeq shall be processed according to Section 8.5.4 on page 152.

  The missing I-frame(s) are considered lost and ExpectedTXSeq is set equal to TxSeq+1 as specified in Section 8.5.2 on page 151. The missing I-frame(s) are reported as lost to the SDU reassembly function.

- *Invalid TxSeq*

  An invalid TxSeq value is a value that does not meet either of the above conditions and TxSeq is not equal to ExpectedTxSeq. An I-frame with an invalid TxSeq is likely to have errors in the control field and shall be silently discarded.

#### *8.5.6.2 ReqSeq Sequence error*

An ReqSeq sequence error exception condition occurs in the transmitter when a valid S-frame or I-frame is received which contains an invalid ReqSeq value. An invalid ReqSeq is one that is not in the range ExpectedAckSeq ≤ ReqSeq ≤ NextTXSeq.

The L2CAP entity shall close the channel as a consequence of an ReqSeq Sequence error.

An L2CAP entity that fails to receive an acknowledgement for an I-frame shall, on the expiry of the RetransmissionTimer, take appropriate recovery action as defined in Section 8.5.5 on page 152.

#### *8.5.6.3 Invalid frame*

Any frame received that is invalid (as defined in Section 3.3.6 on page 52) shall be discarded, and no action shall be taken as a result of that frame, unless the receiving L2CAP entity is configured to deliver erroneous frames to the layer above L2CAP. In that case, the data contained in invalid frames may also be added to the receive buffer and made available for pulling from the SDU reassembly function.

## 8.6 ENHANCED RETRANSMISSION MODE

Enhanced Retransmission mode operates as an HDLC balanced data link operational mode. Either L2CAP entity may send frames at any time without receiving explicit permission from the other L2CAP entity. A transmission may contain single or multiple frames and shall be used for I-frame transfer and/or to indicate status change.

### 8.6.1 Function Of PDU Types

Enhanced Retransmission mode uses I-frames to transfer upper layer information and S-frames for supervision. There are four S-frames defined: Receiver Ready (RR), Reject (REJ), Receiver Not Ready (RNR), and Selective Reject (SREJ). All frames formats in Enhanced Retransmission mode shall use the Enhanced Control Field.

#### *8.6.1.1 Receiver Ready (RR)*

The RR frame shall be used by an L2CAP entity to

1. Indicate that it is ready to receive I-frames
2. Acknowledge previously received I-frames numbered up to ReqSeq –1 inclusive.

An RR with P-bit set to 1 (P=1) is used to indicate the clearance of any busy condition that was initiated by an earlier transmission of an RNR frame by the same L2CAP entity.

#### *8.6.1.2 Reject (REJ)*

The REJ frame shall be used by an L2CAP entity to request retransmission of I-frames starting with the frame numbered ReqSeq. I-frames numbered ReqSeq –1 and below shall be considered acknowledged. Additional I-frames awaiting initial transmission may be transmitted following the retransmitted I-frame(s) up to the TxWindow size of the receiver.

At most only one REJ exception from a given L2CAP entity to another L2CAP entity shall be established at any given time. A REJ frame shall not be transmitted until an earlier REJ exception condition or all earlier SREJ exception conditions have been cleared. The REJ exception condition shall be cleared upon the receipt of an I-frame with TxSeq equal to the ReqSeq of the REJ frame.

Two L2CAP entities may be in REJ exception conditions with each other at the same time.

#### *8.6.1.3 Receiver Not Ready (RNR)*

The RNR frame shall be used by an L2CAP entity to indicate a busy condition (i.e. temporary inability to receive I-frames). I-frames numbered up to ReqSeq –1 inclusive shall be considered acknowledged. The I-frame numbered ReqSeq and any subsequent I-frames sent shall not be considered acknowledged. The acceptance status of these I-frames shall be indicated in subsequent transfers.

#### *8.6.1.4 Selective Reject (SREJ)*

The SREJ frame shall be used by an L2CAP entity to request retransmission of one I-frame. The ReqSeq shall indicate the TxSeq of the earliest I-frame to be retransmitted (not yet reported by a SREJ). If the P-bit is set to 1 then I-frames numbered up to ReqSeq -1 inclusive shall be considered acknowledged. If the P-bit is set to 0 then the ReqSeq field in the SREJ shall not indicate acknowledgement of I-frames.

Each SREJ exception condition shall be cleared upon receipt of an I-frame with TxSeq equal to the ReqSeq sent in the SREJ frame.

An L2CAP entity may transmit one or more SREJ frames with the P=0 before one or more earlier SREJ exception conditions initiated with SREJ(P=0) have been cleared. An L2CAP entity shall not transmit more than one SREJ with P=1 before all earlier SREJ exception conditions have been cleared. A SREJ frame shall not be transmitted if an earlier REJ exception condition has not been cleared. Likewise a REJ frame shall not be transmitted if one or more SREJ exception conditions have not been cleared. Only one I-frame shall be retransmitted in response to receiving a SREJ frame with P= 0. Additional I-frames awaiting initial transmission may be transmitted following the retransmission of the specific I-frame requested by SREJ with P=1.

#### *8.6.1.5 Functions of the Poll (P) and Final (F) bits.*

P-bit set to 1 shall be used to solicit a response frame with the F-bit set to 1 from the remote L2CAP entity at the earliest respond opportunity. At most only one frame with a P=1 shall be outstanding in a given direction at a given time. Before an L2CAP entity issues another frame with P=1, it shall have received a response frame from the remote L2CAP entity with F=1. If no valid frame is received with F=1 within Monitor time-out period, the frame with P=1 may be retransmitted.

The Final bit shall be used to indicate the frame as a response to a soliciting poll (S-frame with P=1). The frame with F=1 shall not be retransmitted. The Monitor-timeout is not used to monitor lost frames with F=1. Additional frames with F=0 may be transmitted following the frame with F=1.

S-frames shall not be transmitted with both the F-bit and the P-bit set to 1 at the same time.

### 8.6.2 Rules For Timers

Timers are started upon transmission of a packet. Timers should be started when the corresponding packet leaves the controller (transmitted or flushed). If the timer is not started when the packet leaves the controller then it shall be started when the packet is delivered to the controller. The specific rules for BR/

EDR Controllers, BR/EDR/LE Controllers, and AMP Controllers are described in the following sections.

### *8.6.2.1 Timer Rules for ACL-U Logical Links*

If a flush timeout does not exist on the ACL-U logical link for the channel using Enhanced Retransmission mode then the value for the Retransmission time-out shall be at least two seconds and the value for the Monitor time-out shall be at least twelve seconds. If a flush timeout exists on the link for Enhanced Retransmission mode then the value for the Retransmission time-out shall be the larger of:

- the value of the flush timeout multiplied by three
- one second

If a flush timeout exists on the link for Retransmission mode and both sides of the link are configured to the same flush timeout value then the monitor time-out shall be set to a value at least as large as the Retransmission time-out otherwise the value of the Monitor time-out shall be the larger of:

- the value of the flush timeout multiplied by six
- twelve seconds

If an L2CAP entity knows that a specific packet has been flushed instead of transmitted then it may execute proper error recovery procedures immediately.

When configuring a channel over an ACL-U logical link the values sent in a Configuration Request packet for Retransmission timeout and Monitor timeout shall be 0.

**Note**: If the link has a flush timeout and the Non-Flushable Packet Boundary Flag feature is used to mark the Enhanced Retransmission mode packets as non-flushable then the link does not have a flush timeout with regards to Enhanced Retransmission mode.

### *8.6.2.2 Timer Rules for AMP Controllers*

AMP Controllers can be classified by their behavior as follows:

1. Attempt to send a packet until LSTO disconnects the physical link.
2. Attempt to send a packet until a Controller based retry counter is exceeded whereupon the packet is flushed.

Class 1 AMP controllers are reliable. They will not lose packets though packets can be lost during a move operation. Class 2 AMP controllers are not reliable and can lose packets. The Best Effort Flush Timeout field in the AMP Info (see [Vol. 2] Part E, Section 7.5.8) can be used to determine the behavior of the controller (i.e. class 1 or class 2). If the Best Effort Flush Timeout field in the

AMP info is 0xFFFFFFFF then the class is 1 otherwise the class is 2. The Best Effort Flush Timeout field indicates the flush timeout that may be set on the Best Effort logical link.

For class 1 AMP controllers the Retransmission and Monitor timers are not really needed. Packets should not be lost. Packets lost during the move operation will be detected by L2CAP and retransmitted as part of the move operation. Therefore, the Retransmission timeout and Monitor timeout shall be at least the value of the Link supervision timeout set on the link or can be turned off altogether. The values sent in a Configuration Request packet for Retransmission timeout and Monitor timeout shall be 0.

When configuring a channel the AMP-U logical link of a over a class 2 AMP Controller non-0 values for Retransmission timeout and Monitor timeout shall be sent in a Configuration Request packet. The non-0 values specify the “processing” time of received packets. Processing time includes the time it takes for a received packet to be passed from the device's Controller to the L2CAP layer plus the time to be processed by the L2CAP layer and a response submitted back to the Controller. The local device's processing time is used by the remote device in calculating the Retransmission timeout and Monitor timeout it will use. The timeout values that will actually be used by a device shall be sent in a Configuration Response packet.

As with BR/EDR and BR/EDR/LE Controllers, the method used for setting timer values for class 2 AMP Controllers depends on when timers are started. Timers are either started when the packet leaves the Controller or when the packet is delivered to the Controller.

The timeout values used for Class 2 AMP Controllers where timers are started when the packet leaves the controller shall be at least the values received in the Configuration Request packet from the remote device (remote device's “processing” time).

For Class 2 AMP Controller where timers are started when the packet is delivered to the Controller the following rules shall be used:

1. The local L2CAP Entity shall set a flush timeout on the Best Effort logical link equal to the value provided by the local Controller in the Best Effort Flush Timeout field of AMP info structure.
2. The value of the Retransmission timeout and Monitor Timeout shall be the value of the flush timeout multiplied by three plus the corresponding processing time received in the Configuration Request packet from the remote device.

If an L2CAP entity knows that a specific packet has been flushed instead of transmitted then it may execute proper error recovery procedures immediately.

#### *8.6.2.3 Timer Values used After a Move Operation*

When a channel is moved from one Controller to another Controller the timeout values used after the move operation are based on the capabilities of the new Controller. The timeout values shall be set according to the following table:

| Controller | Timeout Values |
|---|---|
| BR/EDR and BR/EDR/LE | Retransmission timeout – at least 2 seconds<br>Monitor timeout – at least 12 seconds |
| Class 1 AMP Controller | Retransmission timeout – at least LSTO (or turned off)<br>Monitor timeout – at least LSTO (or turned off) |
| Class 2 AMP Controller where timers are started when packets leave the controller | Retransmission timeout – at least 500ms<br>Monitor timeout – at least 500ms<br>Note: 500ms is the default value for the “processing” time of received packets. |
| Class 2 AMP Controller where timers are started when packets are delivered to the controller | Retransmission timeout – at least (local controller "Best Effort” Flush Timeout * 3) + 500ms<br>Monitor timeout – at least (local controller “Best Effort” Flush Timeout * 3) + 500ms<br>Note: 500ms is the default value for the “processing” time of received packets |

*Table 8.2: AMP Controller timeout values*

When moving to a Class 2 AMP Controller, the Retransmission Timeout and Monitor timeout should be reconfigured after the move operation to get the actual value for the “processing” time of received packets.

### 8.6.3 General Rules for the State Machine

Enhanced Retransmission mode is specified using a pair of state machines, a Transmitter state machine and a Receiver state machine. The following rules apply to the state machine pair.

1. The state machine pair is informative but described using normative text in order to clearly specify the behavior of the protocol. Designers and implementers may choose any design / implementation technique they wish, but it shall behave in a manner identical to the external behavior of the specified state machines.
2. There is a single state machine pair for each active L2CAP channel configured to use Enhanced Retransmission mode.
3. Variables are used to limit the number of states by maintaining the state of particular conditions. The variables are defined in section 8.6.5.2.
4. For some combinations of Event and Condition, the state tables provide alternative groups of actions. These alternatives are separated by horizontal

lines in the Actions and Next State columns. The alternatives are mutually exclusive; selection of an alternate is done based upon (i) local status, (ii) a layer management action, or (iii) an implementation decision. There is no relationship between the order of the alternatives between events, nor is it implied that the same alternative must be selected every time the event occurs.

5. The state tables use timers. Any Start Timer action restarts the specified timer from its initial value, even if the timer is already running. When the timer reaches 0 the appropriate timer expired event is set and the timer stops. The Stop Timer action stops a timer if it is running.
6. Events not recognized in a particular state are assumed to remain pending until any masking flag is modified or a transition is made to a state where they can be recognized.
7. Some state transitions and actions are triggered by internal events (e.g. requests from the upper layer). It is implementation specific how these internal events are realized. They are used for clarity in specifying the state machine. All events including Internal events are described in Section 8.6.5.3 on page 162.
8. The state machines specify the exact frames to be sent by transmitters but are relaxed on what receivers are allowed to accept as valid. For example there are cases where the transmitter is required to send a frame with P=1. The correct response is a frame with F=1 but in some cases the receiver is allowed to accept a frame with F=0 in addition to F=1.

### 8.6.4 State Diagram

The state diagram shows the states and the main transitions. Not all events are shown on the state diagram.

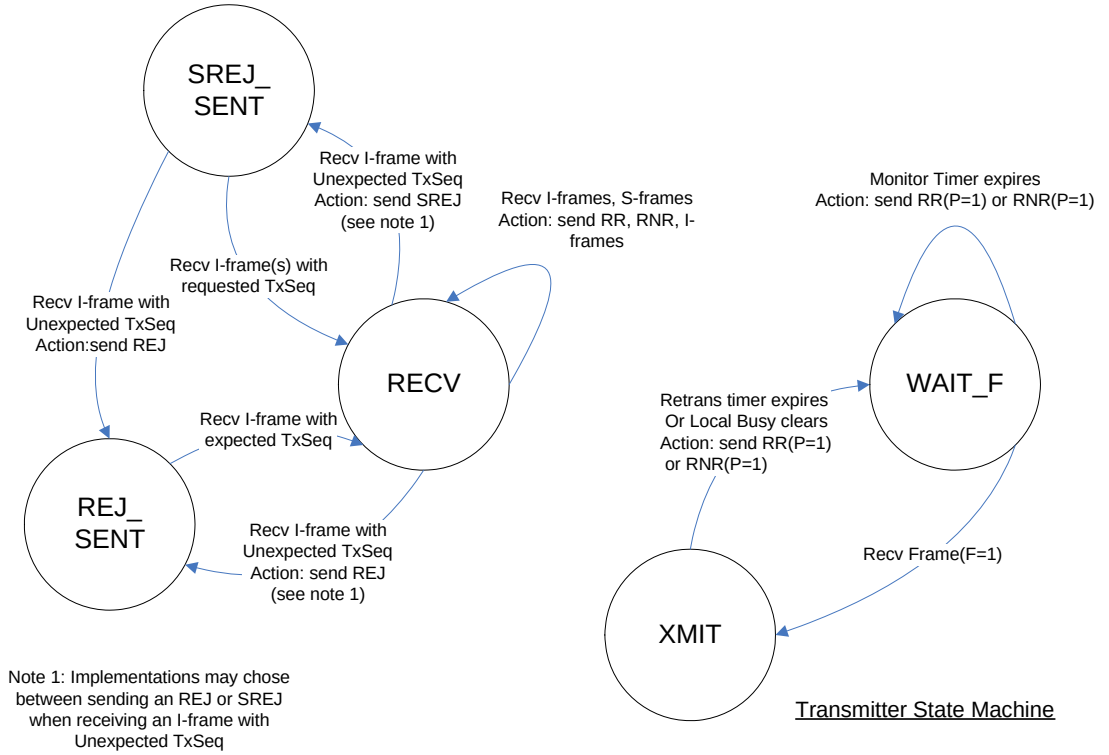



*Figure 8.4: Receiver state machine*

### 8.6.5 States Tables

#### *8.6.5.1 State Machines*

Enhanced Retransmission mode is described as a pair of state machine. The Receiver state machine handles all received frames while the Transmitter State machine handles all asynchronous events including requests from the upper layer and the expiration of timers.

The Receiver state machine “calls” the Transmitter state machine using the PassToTx action. This shows up in the Transmitter state machine as an event. When the Transmitter state machine is called it runs to completion before returning to the Receiver state machine. Running to completion means that all actions are executed and the Transmitter state is changed to the new state.

The Receiver and Transmitter state machine share variables and timers.

#### *8.6.5.2 States*

The following states have been defined to specify the protocol; the actual number of states and naming in a given implementation is outside the scope of this specification:

**RECV**—This is the main state of the Receiver state machine.

**REJ_SENT**—The L2CAP entity has sent a REJ frame to cause the remote L2CAP entity to resend I-frame(s). The L2CAP entity is waiting for the I-frame with a TxSeq that matches the ReqSeq sent in the REJ. Whether to send a REJ versus a SREJ is implementation dependent.

**SREJ_SENT**—The L2CAP entity has sent one or more SREJ frames to cause the remote L2CAP entity to resend missing I-frame(s). The local L2CAP entity is waiting for all requested I-frames to be received. If additional missing I-frames are detected while in SREJ_SENT then additional SREJ frames or a REJ frame can be sent to request those I-frames. Whether to send a SREJ versus a REJ is implementation dependent.

**XMIT**—This is the main state of the Transmitter state machine.

**WAIT_F**—Local busy has been cleared or the Retransmission timer has expired and an S-frame with P=1 has been sent. The local L2CAP entity is waiting for a frame with F=1. I-frames cannot be sent while in the WAIT_F state to prevent the situation where retransmission of I-frames could result in the channel being disconnected.

#### *8.6.5.3 Variables and Timers*

Variables are used to limit the number of states and help clarify the state chart tables. Variables can be set to values, evaluated in conditions and compared in conditional statements. They are also used in the action descriptions. Below is a list of the operators, connectives and statements that can be used with variables.

| Operator, connective or statement | Description |
|---|---|
| := | Assignment operator. Used to set a variable to a value |
| = | Relational operator "equal" |
| > | Relational operator "greater than." |
| < | Relational operator "less than" |
| ≥ | Relational operator "greater than or equal" |
| ≤ | Relational operator "less than or equal" |
| + | Arithmetic operator "plus" |
| mod | Modulo operator - returns the remainder of division of one number by another. |
| **and** | logical connective "and." It returns TRUE if both operands are TRUE otherwise it returns FALSE. |

*Table 8.3: Enhanced Retransmission mode uses the following variables and sequence numbers described in Section 8.3*

| Operator, connective or statement | Description |
|---|---|
| **or** | logical connective "or." It returns TRUE if either of its operands are TRUE otherwise it returns FALSE. |
| **if** (expression) **then**<br>statement | Conditional Statement. If expression is TRUE then the statement is executed otherwise the statement is not executed. The statement is composed of one or more actions. All the actions in the statement are indented under the **if** … **then** clause. |
| **if** (expression) **then**<br>statement1<br>**else**<br>statement2 | Conditional Statement. If expression is TRUE then statement1 is executed otherwise statement2 is executed. A statement is composed of one or more actions. All the actions in the statement1 are indented under the **if** … **then** clause and all the actions of statement2 are indented under the else clause. |

*Table 8.3: Enhanced Retransmission mode uses the following variables and sequence numbers described in Section 8.3*

- TxSeq
- NextTxSeq
- ExpectedAckSeq
- ReqSeq
- ExpectedTxSeq
- BufferSeq

In addition to the variables above the following variables and timers are used.

**RemoteBusy**—when set to TRUE RemoteBusy indicates that the local L2CAP entity has received an RNR from the remote L2CAP entity and considers the remote L2CAP entity as busy. When the remote device is busy it will likely discard I-frames sent to it. The RemoteBusy flag is set to FALSE when the local L2CAP Entity receives an RR, REJ or SREJ. When set to FALSE the local L2CAP entity considers the remote L2CAP entity able to accept I-frames. When the channel is created RemoteBusy shall be set to FALSE.

**LocalBusy**—when set to TRUE, LocalBusy indicates the local L2CAP entity is busy and will discard received I-frames. When set to FALSE the local L2CAP entity is not busy and is able to receive I-frames. When the channel is created LocalBusy shall be set to FALSE.

**UnackedFrames**—holds the number of unacknowledged I-frames. When the channel is created UnackedFrames shall be set to 0.

**UnackedList**—holds the unacknowledged I-frames so they can be retransmitted if necessary. I-frames in the list are accessed via their TxSeq number. For example UnackedList[5] accesses the I-frame with TxSeq 5.

**PendingFrames**—holds the number of pending I-frames. I-frames passed to L2CAP from the upper layer may not be able to be sent immediately because the remote L2CAP entity's TxWindow is full, is in a busy condition or the local L2CAP is in the incorrect state. When I-frames cannot be sent they are stored in a queue until conditions allow them to be sent. When the channel is created PendingFrames shall be set to 0.

**SrejList**—is a list of TxSeq values for I-frames that are missing and need to be retransmitted using SREJ. A SREJ has already been sent for each TxSeq on the list. When SrejList is empty it equals 0 (i.e. SrejList = 0). If SrejList is not empty it is greater than 0 (i.e. SrejList > 0).

**RetryCount**—holds the number of times an S-frame operation is retried. If an operation is tried MaxTransmit times without success the channel shall be closed.

**RetryIframes**[]—holds a retry counter for each I-frame that is sent within the receiving device's TxWindow. Each time an I-frame is retransmitted the corresponding counter within RetryIframes is incremented. When an attempt to retransmit the I-frame is made and the counter is equal to MaxTransmit then the channel shall be closed.

**RNRsent**—when set to TRUE it means that the local L2CAP entity has sent an RNR frame. It is used to determine if the L2CAP entity needs to send an RR to the remote L2CAP entity to clear the busy condition. When the channel is created RNRsent shall be set to FALSE.

**RejActioned**—is used to prohibit a frame with F=1 from causing I-frames already retransmitted in response to a REJ from being retransmitted again. RejActioned is set to TRUE if a received REJ is actioned when a frame sent with P=1 is unanswered. When the channel is created RejActioned shall be set to FALSE.

**SrejActioned**—is used in conjunction with SrejSaveReqSeq to prohibit a frame with F=1 from causing an I-frame already retransmitted in response to a SREJ from being retransmitted again. SrejActioned is set to TRUE if a received SREJ is actioned when a frame sent with P=1 is unanswered. When the channel is created SrejActioned shall be set to FALSE.

**SrejSaveReqSeq**—is used to save the ReqSeq of a SREJ frame that causes SrejActioned to be set to TRUE.

**SendRej**—when set to TRUE it indicates that the local L2CAP entity has determined that a REJ should be sent in the SREJ_SENT state while processing received I-frames. The sending of new SREJ frames is stopped. When the channel is created SendRej shall be set to FALSE.

**BufferSeqSrej**—is used while in the SREJ_SENT state to keep track of the value to which BufferSeq will be set upon exit of the SREJ_SENT state.

**FramesSent**—is used to keep track of the number I-frames sent by the Send-Data and Retransmit-I-frames actions.

**MaxTxWin**—contains the maximum window size plus 1. It is used in TxWindow modulo operations as the divisor and in state table conditions. This value shall be set to 16384 (0x4000) if the Extended Window Size option is used; otherwise it shall be set to 64.

**RetransTimer**—The Retransmission Timer is used to detect lost I-frames. When the channel is created the RetransTimer shall be off.

**MonitorTimer**—The Monitor Timer is used to detect lost S-frames. When the channel is created the MonitorTimer shall be off.

#### *8.6.5.4 Events*

**Data-Request**—The upper layer has requested that an SDU be sent. The SDU may need to be broken into multiple I-frames by L2CAP based on the MPS of the remote device and/or the maximum PDU allowed by the HCI or QoS requirements of the system.

**Local-Busy-Detected**—A local busy condition occurs when the local L2CAP entity is temporarily unable to receive, or unable to continue to receive, I-frames due to internal constraints. For example, the upper layer has not pulled received I-frames and the local L2CAP entity needs to send an acknowledgment to the remote L2CAP entity. The method for handling the detection of local busy is implementation specific. An implementation may wait to send an RNR to see if the busy condition will clear before the remote L2CAP entity's Retransmission timer expires. If the busy condition clears then frames can be acknowledged with an RR or I-frame. If the busy condition does not clear before the remote L2CAP entity's Retransmission timer expires then an RNR shall be sent in response to the RR or RNR poll sent by the remote L2CAP entity. Optionally an implementation may send an RNR as soon as the local busy condition is detected.

**Local-Busy-Clear**—The local busy condition clears when L2CAP has buffer space to receive more I-frames (i.e. SDU Reassembly function and/or upper layer has pulled I-frames) and if necessary the upper layer has cleared the busy condition.

**Recv ReqSeqAndFbit**—This is an event generated by the Receiver state machine. It contains the ReqSeq and F-bit value of a received frame. The value of the F-bit can be checked in a condition.

**Recv Fbit**—This is an event generated by the Receiver state machine. It contains the F-bit value of a received frame. The value of the F-bit can be checked in a condition.

**RetransTimer-Expires**—The Retransmission Timer has counted to down to 0 and stopped.

**MonitorTimer-Expires**—The Monitor Timer has counted down to 0 and stopped.

**Recv I-frame**—Receive an I-frame with any value for the F-bit.

**Recv RR, REJ, RNR, SREJ (P=x)** or **(F=x)**—Receive a specific S-frame (RR, REJ, etc.) with a specific value for the P and/or F bit. The F-bit and the P-bit shall not both be set to 1 in a transmitted S-frame so received S-frames with both P and F set to 1 should be ignored. If the P and/or F bit value is not specified in the event then either value is accepted.

**Recv RROrRNR**—Receive an RR or RNR with any value for the P-bit and F-bit.

**Recv REJorSREJ**—Receive an REJ or SREJ with any value for the P-bit and F-bit.

**Recv frame**—This is catch-all for all frames that are not explicitly declared as events in the state table.

#### *8.6.5.5 Conditions*

**RemoteBusy = TRUE or FALSE**—TRUE indicates the remote L2CAP entity is in a busy condition and FALSE indicates the remote L2CAP entity is not busy.

**LocalBusy = TRUE or FALSE**—TRUE indicates the local L2CAP entity is in a busy condition and FALSE indicates the local L2CAP entity is not busy.

**RemWindow-Not-Full**—The number of unacknowledged I-frames sent by L2CAP has not yet reached the TxWindow size of the remote L2CAP entity.

**RemWindow-Full**—The number of unacknowledged I-frames sent by the L2CAP entity has reached the TxWindow size of the remote L2CAP entity. No more I-frames shall be sent until one or more I-frames have been acknowledged.

**RNRsent = TRUE or FALSE**—**TRUE** indicates an RNR has been sent while a local busy condition exists. It is set to FALSE when the local busy condition clears.

**F = 0 or 1**—the F-bit of a received frame is checked. The F-bit of the received frame is available as part of the Recv ReqSeqAndFbit and Recv Fbit events.

**RetryIframes[i] < or ≥ MaxTransmit**—Compare the appropriate counter in RetryIframes after processing the ReqSeq in the receive frame to determine if it has reached MaxTransmit or not.

**RetryCount < or ≥ MaxTransmit**—Compare RetryCount to determine if it has reached MaxTransmit or not.

**With-Expected-TxSeq**—The TxSeq of a received I-frame is equal to ExpectedTxSeq.

**With-Valid-ReqSeq**—The ReqSeq of the received frame is in the range ExpectedAckSeq ≤ ReqSeq < NextTxSeq.

**With-Valid-ReqSeq-Retrans**—The ReqSeq of the received frame is in the range ExpectedAckSeq ≤ ReqSeq < NextTxSeq.

**With-Valid-F-bit** —The F-bit of a received frame is valid if it is 0 or if it is 1 and a frame sent with P=1 by the local L2CAP entity is unanswered (i.e. the local L2CAP entity send a frame with P=1 and has not yet received a frame with F=1 until receiving this one). If the Transmitter state machine is in the WAIT_F state then a frame sent with P=1 is unanswered.

**With-unexpected-TxSeq**—The TxSeq of the received I-frame is within the TxWindow of the L2CAP entity receiving the I-frame but has a TxSeq “greater” than ExpectedTxSeq where “greater” means later in sequence than ExpectedTxSeq.

**With-duplicate-TxSeq**—The TxSeq of the received I-frame is within the TxWindow of the L2CAP entity receiving the I-frame but has a TxSeq “less” than ExpectedTxSeq where “less” means earlier in the sequence than ExpectedTxSeq. In other words this is a frame that has already been received.

**With-Invalid-TxSeq**—The TxSeq of the received I-frame is not within the TxWindow of the L2CAP entity receiving the frame.

**With-Invalid-ReqSeq**—The ReqSeq of the received frame is not in the range ExpectedAckSeq ≤ ReqSeq < NextTxSeq.

**With-Invalid-ReqSeq-Retrans**—The ReqSeq of the received frame is not in the range ExpectedAckSeq ≤ ReqSeq < NextTxSeq.

**Not-With-Expected-TxSeq**—The TxSeq of the received I-frame is within the TxWindow of the L2CAP entity receiving the frame but is not equal to ExpectedTxSeq. It is either unexpected or a duplicate.

**With-Expected-TxSeq-Srej**—The TxSeq of the received I-frame is equal to the TxSeq at the head of SrejList.

**SendRej = TRUE or FALSE**—**TRUE** indicates that a REJ will be sent after all frames requested using SREJ have been received.

**SrejList = or > 1**—Determine if the number of items in SrejList is equal to or greater than 1.

**With-Unexpected-TxSeq-Srej**—The TxSeq of the received I-frame is equal to one of the values stored in SrejList but is not the TxSeq at the head. This indicates that one or more I-frames requested using SREJ are missing either because the SREJ was lost or the requested I-frame(s) were lost. Either way the SREJ frames must be resent to retrieve the missing I-frames.

**With-duplicate-TxSeq-Srej**—The TxSeq of the received I-frame is equal to a TxSeq of one of the saved I-frames indicating it is a duplicate.

#### *8.6.5.6 Actions*

**Send-Data**—This action is executed as a result of a Data-Request event. The number of I-frames sent without being acknowledged shall not exceed the TxWindow size of the receiving L2CAP entity (UnackedFrames is less than or equal to the remote L2CAP entity's TxWindow). Any I-frames that cannot be sent because they would exceed the TxWindow size are queued for later transmission. For each I-frame the following actions shall be carried out:

Send I-frame with TxSeq set to NextTxSeq and ReqSeq set to BufferSeq.

UnackedList[NextTxSeq] := I-frame

UnackedFrames := UnackedFrames + 1

FramesSent := FramesSent + 1

RetryIframes[NextTxSeq] := 1

NextTxSeq := (NextTxSeq + 1) mod MaxTxWin

Start-RetransTimer

**Pend-Data**—This action is executed as a result of a Data-Request when it is not possible to send I-frames because the window is full, the remote L2CAP entity is in a busy condition or the local L2CAP entity is not in a state where I-frames can be sent (e.g. WAIT_F). The I-frame(s) are queued for later transmission.

**Process-ReqSeq**—the ReqSeq contained in the received frame shall acknowledge previously transmitted I-frames. ExpectedAckSeq shall be set to ReqSeq to indicate that the I-frames with TxSeq up to and including (ReqSeq—1) have been acknowledged. The acknowledged I-frames shall be removed from UnackedList, the retry counters for each acknowledged frame shall be set to 0 and the number of acknowledged frames shall be subtracted from UnackedFrames so that UnackedFrames shall contain the number of the remaining unacknowledged I-frames. Pending I-frames are now available to be transmitted by the Send-Ack action. If UnackedFrames equals 0 then Stop-RetransTimer.

**Send RR, RNR (P=x) or (F=x)**—Send the specified S-frame with the specified value for the P-bit or F-bit. If a value for the P-bit or F-bit is not specified the value shall be 0. For example Send RR(P=1) means send an RR with the P-bit

set to 1 and the F-bit set to 0. The ReqSeq field shall be set to BufferSeq. If an RNR is sent, RNRsent shall be set to TRUE.

**Send REJ (P =x) or (F=x)**—Send a REJ with the specified value for the P-bit or F-bit. The ReqSeq field shall be set to ExpectedTxSeq. If a value for the P-bit or F-bit is not specified the value shall be 0. Note that this will acknowledge previously received I-frames up to ExpectedTxSeq—1 and may allow the remote L2CAP entity to transmit new I-frames. If the local L2CAP entity is not in a position to acknowledge the previously received I-frames it may use SREJ(P=0) or RNR. It may also wait to send the REJ until it is able to acknowledge the I-frames.

**Send RRorRNR (P=x) or (F=1)**—Send an RR or RNR with the specified value for the P-bit or F-bit based on the value of LocalBusy. If a value for the P-bit or F-bit is not specified the value shall be 0. An RNR shall be sent if LocalBusy equals TRUE. If LocalBusy equals FALSE then an RR shall be sent.

**Send IorRRorRNR(F=1)**—Send I-frames, an RR or an RNR with the F-bit set to 1. The following algorithm shall be used:

```
FramesSent:=0
    if LocalBusy = TRUE then
        Send RNR(F=1)
    if RemoteBusy = TRUE and UnackedFrames > 0 then
        Start-RetransTimer
    Send-Pending-I-frames(see note)
    if LocalBusy = FALSE and FramesSent = 0 then
         Send RR(F=1)
```

Note: The SendIorRRorRNR(F=1) sends frames by invoking other actions. During the execution of SendIorRRorRNR multiple actions may be invoked. The first action invoked shall send the first or only frame with the F-bit set to 1. All other frames sent shall have the F-bit set to 0.

**Send SREJ**—Send one or more SREJ frames with P=0. For each missing I-frame starting with ExpectedTxSeq up to but not including the TxSeq of the received I-frame, an SREJ frame is sent with ReqSeq set to the TxSeq of the missing frame. The TxSeq is inserted into the tail of SrejList. For example if ExpectedTxSeq is 3 and the received I-frame has a TxSeq of 5 there are two missing I-frames. An SREJ with ReqSeq 3 is sent followed by an SREJ with ReqSeq 4. TxSeq 3 is inserted first into SrejList followed by TxSeq 4. After all SREJ frames have been sent ExpectedTxSeq shall be set to the TxSeq of the received I-frame + 1 mod MaxTxWin.

**Send SREJ(SrejList)**—Send one or more SREJ frames with P=0. An I-frame was received that matches one of the TxSeq values in the SrejList but does not match the head of SrejList. This means I-frames requested via SREJ are still

missing. For each TxSeq value starting with the head of SrejList and going backwards (i.e., from the head towards the tail) through the SrejList up to but not including the TxSeq of the received frame, an SREJ frame is sent with ReqSeq set to the TxSeq from SrejList. The TxSeq is removed from SrejList and reinserted into the tail of SrejList. Finally, remove the TxSeq of the received frame from the head of the list.

**Send SREJ(SrejList-tail)(F=1)**—Send a SREJ frame with F=1 and ReqSeq equal to the TxSeq at the tail of SrejList.

**Start-RetransTimer**—If the Monitor timer is not running then start the Retransmission Timer from its initial value (see Retransmission time-out in Section 5.4). If the Retransmission timer is already running it is restarted from its initial value. If the Monitor timer is running then the Retransmission timer is not started.

**Start-MonitorTimer**—Start the Monitor Timer from its initial value (see Monitor time-out in Section 5.4 on page 86). If the timer is already running it is restarted from its initial value.

**PassToTx**—Pass the ReqSeq and F-bit value of a received frame to the Transmitter state machine. This will show up as a Recv ReqSeqAndFbit event in the Transmitter state machine.

**PassToTxFbit**—Pass the F-bit value of a received frame to the Transmitter state machine. This will show up as a Recv Fbit event in the Transmitter state machine.

**Data-Indication**—A received I-frame is passed to the SDU reassembly function. For the purpose of the state machine this operation is completed immediately so the Send_Ack action should be executed as one of the next actions. In some cases the SDU reassembly function cannot accept the I-frame so the I-frame will be stored within the L2CAP Entity consuming a portion of its TxWindow. When the I-frame is pulled by the SDU reassembly function the Send_Ack action should be executed. Before the Send_Ack action is executed BufferSeq is advanced as follows:

BufferSeq := (BufferSeq + 1) mod MaxTxWin

**Increment-ExpectedTxSeq**—ExpectedTxSeq is incremented as follows:

ExpectedTxSeq := (ExpectedTxSeq + 1) mod MaxTxWin

**Stop-RetransTimer**—the Retransmission timer is stopped.

**Stop-MonitorTimer** —the Monitor timer is stopped.

**Send-Ack (F=x)**—an acknowledgement with the specified value for the F-bit may be sent. Note that this action may occur in an action block with other actions that also send frames. If a frame has already been sent then it is not necessary to send additional frames. If the value for the F-bit is not specified it shall be set to 0. If the value specified is P then the F-bit shall be set equal to the value of the P-bit of the received frame being acknowledged. If more than

one frame is sent in the acknowledgment only the first frame shall have an F-bit set to 1. An acknowledgement is an RR, RNR, or pending I-frame(s) (I-frames that have not been transmitted yet). If pending I-frames are available and are allowed to be sent then as many as allowed should be sent as an acknowledgment. Sending an RR or RNR as an acknowledgment for each received I-frame is not required. An implementation may wait to send an RR or RNR until a specific number of I-frames have been received, after a certain period of time has elapsed or some other algorithm. To keep data flowing it is recommended that an acknowledgment be sent before the TxWindow is full. It should also be noted that the maximum size of a remote L2CAP entity's unacknowledged I-frame list may be smaller than the local L2CAP entity's TxWindow. Therefore the local L2CAP entity should not expect the remote L2CAP entity to send enough frames to fill its TxWindow and should acknowledge I-frames accordingly. The following algorithm shall be used when sending an acknowledgment.

```
if LocalBusy == TRUE then
Send_RNR(F=x)
else if (RemoteBusy == FALSE) and Pending I-frames
Exist and RemWindow-Not-Full
Send-Pending-I-frames (F=x)
else
Send_RR (F=x)
```

**InitSrej**—Initialize the variables used for processing SREJ as follows:

Clear SrejList—(remove all values)

SendRej := FALSE

BufferSeqSrej := BufferSeq

**SaveIframeSrej**—Save the received I-frame. Missing I-frame(s) will be retransmitted in response to SREJ frames. Implementations may want to save the I-frame in its proper sequence order by leaving room for the missing I-frames.

**StoreOrIgnore**—If the local L2CAP entity has room to store the received I-frame then it may store it otherwise it shall discard it. If the received I-frame is stored, ExpectedTxSeq is advanced as follows:

```
ExpectedTxSeq := (ExpectedTxSeq + 1) mod MaxTxWin
```

**PbitOutstanding**—If the Transmitter state machine of the local L2CAP entity is in the WAIT_F state then return TRUE otherwise return FALSE.

**Retransmit-I-frames**—All the unacknowledged I-frames starting with the I-frame with TxSeq equal to the ReqSeq field of the received S-frame (REJ or RR) is retransmitted. If the P-bit of the received S-frame is 1 then the F-bit of the first I-frame sent shall be 1. If the P-bit of the received S-frame is 0 then the F-bit of the first I-frame sent shall be 0. The F-bit of all other unacknowledged I-frames sent shall be 0. The retry counter in RetryIframes[] for each retransmit-

ted I-frame is incremented by 1. If a retry counter in RetryIframes[] is equal to MaxTransmit then the channel shall be closed. FramesSent shall be incremented by 1 for each frame sent. If the RetransTimer is not already running then perform the Start-RetransTimer action.

**Retransmit-Requested-I-frame**—The unacknowledged I-frame with TxSeq equal to the ReqSeq field of the received S-frame (SREJ) is retransmitted. If the P-bit of the received S-frame is 1 then the F-bit of the retransmitted I-frame shall be 1. If the P-bit of the received S-frame is 0 then the F-bit of the retransmitted I-frame shall be 0. The retry counter in RetryIframes[] corresponding to the retransmitted I-frame is incremented by 1. If the RetransTimer is not already running then perform the Start-RetransTimer action.

**Send-Pending-I-frames (F=x)**—send all pending I-frames that can be sent without exceeding the receiver's TxWindow using the Send-Data action. If a value for the F-bit is specified then the F-bit of the first I-frame sent shall be set to the specified value and the F-bit of all other I-frames sent shall be set to 0. If no value for the F-bit is specified then all I-frames sent shall have the F-bit set to 0. Pending I-frames are I-frames that have been given to the L2CAP entity by the upper layer but have not yet been transmitted. If one or more I-frames are sent and the RetransTimer is not already running then perform the Start-RetransTimer action.

**Close Channel**—Close the L2CAP channel as described in Section 4.6.

**Ignore**—the event may be silently discarded.

**PopSrejList**—Remove and discard the TxSeq from the head of SrejList.

**Data-IndicationSrej**—If the received I-frame fills a gap in a sequence of saved I-frames then all the saved I-frames in the sequence are passed to the SDU reassembly function. For the purpose of the state machine this operation is completed immediately. For example if the TxSeq of saved I-frames before receiving an I-frame is 2, 3, 5, 6, 9 and the received I-frame has a TxSeq of 4 then it fills the gap between 3 and 5 so the sequence 2, 3, 4, 5, 6 can be passed to the SDU reassembly function. When the I-frames are actually removed from the L2CAP entity receive buffers either by being processed immediately or when pulled by the SDU reassembly function, BufferSeqSrej is advanced as follows:

BufferSeqSrej := (BufferSeqSrej + 1) mod MaxTxWin

### 8.6.5.7 XMIT State Table

| Event | Condition | Action | Next State |
|---|---|---|---|
| Data-Request | RemoteBusy = FALSE **and** *RemWindow-Not-Full* | Send-Data | XMIT |
| Data-Request | RemoteBusy = TRUE **or** RemWindow-Full | Pend-Data | XMIT |
| Local-Busy-Detected | | LocalBusy := TRUE | XMIT |
| | | LocalBusy := TRUE<br>Send RNR | XMIT |
| Local-Busy-Clear | RNRsent = TRUE | LocalBusy := TRUE<br>RNRsent = FALSE<br>Send RR(P=1)<br>RetryCount := 1<br>Stop-RetransTimer<br>Start-MonitorTimer | WAIT_F |
| Local-Busy-Clear | RNRsent = FALSE | LocalBusy:= FALSE<br>RNRsent = FALSE | XMIT |
| Recv ReqSeqAndFbit | | Process-ReqSeq | XMIT |
| Recv Fbit | | | XMIT |
| RetransTimer-Expires | | Send RROrRNR(P=1)<br>RetryCount := 1<br>Start-MonitorTimer | WAIT_F |

*Table 8.4: XMIT state table*

### 8.6.5.8 WAIT_F State Table

| Event | Condition | Action | Next State |
|---|---|---|---|
| Data-Request | | Pend-Data | WAIT_F |
| Recv ReqSeqAndFbit | F = 1 | Process-ReqSeq<br>Stop-MonitorTimer<br>**If** UnackedFrames > 0 **then**<br>Start-RetransTimer | XMIT |
| Recv ReqSeqAndFbit | F = 0 | Process-ReqSeq | WAIT_F |
| Recv Fbit | F = 1 | Stop-MonitorTimer<br>**If** UnackedFrames > 0 **then**<br>Start-RetransTimer | XMIT |

*Table 8.5: WAIT_F State Table*

| Event | Condition | Action | Next State |
|---|---|---|---|
| Recv Fbit | F = 0 | | WAIT_F |
| MonitorTimer-Expires | RetryCount < Max-Transmit | RetryCount := Retry-Count+1<br>Send RRorRNR(P=1)<br>Start-MonitorTimer | WAIT_F |
| MonitorTimer-Expires | RetryCount ≥ Max-Transmit | Close Channel | |

*Table 8.5: WAIT_F State Table*

### 8.6.5.9 RECV State Table

| ***Event*** | **Condition** | **Action** | **Next State** |
|---|---|---|---|
| Recv I-frame (F=0) | *With-Expected-TxSeq* **and** *With-Valid-ReqSeq* **and** *With-Valid-F-bit* **and** LocalBusy = FALSE | Increment-ExpectedTxSeq<br>PassToTx<br>Data-Indication<br>Send-Ack(F=0) | RECV |
| Recv I-frame (F=1) | *With-Expected-TxSeq* **and** *With-Valid-ReqSeq* **and** *With-Valid-F-bit* **and** LocalBusy = FALSE | Increment-ExpectedTxSeq<br>PassToTx<br>Data-Indication<br>**If** RejActioned = FALSE **then**<br>Retransmit-I-frames<br>Send-Pending-I-frames<br>else<br>RejActioned := FALSE<br>Send-Ack(F=0) | RECV |
| Recv I-frame | With-duplicate-TxSeq **and** With-Valid-ReqSeq **and** With-Valid-F-bit **and** LocalBusy = FALSE | PassToTx | RECV |
| Recv I-frame | With-unexpected-TxSeq **and** With-Valid-ReqSeq **and** With-Valid-F-bit **and** LocalBusy = FALSE | PassToTx<br>SendREJ | REJ_SENT |
| | | PassToTx<br>InitSrej<br>SaveIframeSrej<br>SendSREJ | SREJ_SENT |
| Recv I-frame | With-Expected-TxSeq **and** With-Valid-ReqSeq **and** With-Valid-F-bit **and** LocalBusy = TRUE | PassToTx<br>StoreOrIgnore | RECV |
| Recv I-frame | With-Valid-ReqSeq **and** Not-With_Expected_TxSeq **and** With-Valid-F-bit **and** LocalBusy = TRUE | PassToTx | RECV |
| Recv RNR (P=0) | With-Valid-ReqSeq **and** With-Valid-F-bit | RemoteBusy := TRUE<br>PassToTx<br>Stop-RetransTimer | RECV |
| Recv RNR (P=1) | With-Valid-ReqSeq **and** With-Valid-F-bit | RemoteBusy := TRUE<br>PassToTx<br>Stop-RetransTimer<br>Send RRorRNR (F=1) | RECV |

*Table 8.6: RECV_State table*

| ***Event*** | **Condition** | **Action** | **Next State** |
|---|---|---|---|
| Recv RR(P=0)(F=0) | With-Valid-ReqSeq **and** With-Valid-F-bit | PassToTx<br>**If** RemoteBusy = TRUE **and** UnackedFrames > 0<br>then<br>Start-RetransTimer<br>RemoteBusy := FALSE<br>*Send-Pending-I-frames* | RECV |
| Recv RR(F=1) | With-Valid-ReqSeq **and** With-Valid-F-bit | RemoteBusy := FALSE<br>PassToTx<br>**If** RejActioned = FALSE **then**<br>Retransmit-I-frames<br>else<br>RejActioned := FALSE<br>Send-Pending-I-frames | RECV |
| Recv RR(P=1) | With-Valid-ReqSeq **and** With-Valid-F-bit | PassToTx<br>*Send IorRRorRNR(F=1*)<br>RemoteBusy := FALSE | RECV |
| Recv REJ (F=0) | With-Valid-ReqSeq-Retrans **and** RetryIframes[i] < Max-Transmit<br>**and** With-Valid-F-bit | RemoteBusy := FALSE<br>PassToTx<br>*Retransmit-I-frames*<br>*Send-Pending-I-frames*<br>If *PbitOutstanding* **then**<br>*RejActioned* := TRUE | RECV |
| Recv REJ (F=1) | With-Valid-ReqSeq-Retrans **and** RetryIframes[i] < Max-Transmit<br>**and** With-Valid-F-bit | RemoteBusy := FALSE<br>PassToTx<br>**If** RejActioned = FALSE **then**<br>Retransmit-I-frameselse<br>RejActioned :=FALSE<br>Send-Pending-I-frames ] | RECV |
| Recv SREJ (P=0) (F=0) | With-Valid-ReqSeq-Retrans **and**<br>RetryIframes[i] < Max-Transmit<br>**and** With-Valid-F-bit | RemoteBusy := FALSE<br>PassToTxFbit<br>Retransmit-Requested-I-frame<br>**If** *PbitOutstanding* **then**<br>SrejActioned := TRUE<br>*SrejSaveReqSeq* = *ReqSeq* | RECV |

*Table 8.6: RECV_State table*

| ***Event*** | **Condition** | **Action** | **Next State** |
|---|---|---|---|
| Recv SREJ (P=0) (F=1) | With-Valid-ReqSeq-Retrans **and**<br>RetryIframes[i] < Max-Transmit<br>**and** With-Valid-F-bit | RemoteBusy := FALSE<br>PassToTxFbit<br>**If** SrejActioned := TRUE **and** SrejSaveReqSeq = ReqSeq<br>then<br>SrejActioned := FALSE<br>else<br>Retransmit-Requested-I-frame | RECV |
| Recv SREJ(P=1) | With-Valid-ReqSeq-Retrans **and**<br>RetryIframes[i] < Max-Transmit<br>**and** With-Valid-F-bit | RemoteBusy := FALSE<br>*PassToTx*<br>*Retransmit-Requested-I-frame*<br>*Send-Pending-I-frames*<br>**If** *PbitOutstanding* **then**<br>SrejActioned = TRUE<br>*SrejSaveReqSeq := ReqSeq* | RECV |
| Recv REJ | With-Valid-ReqSeq-Retrans **and**<br>RetryIframes[i] ≥ Max-Transmit | Close Channel | |
| RECV SREJ | With-Valid-ReqSeq-Retrans **and**<br>RetryIframes[i] >= Max-Transmit | Close Channel | |
| Recv I-frame | (With-Invalid-TxSeq **and** TxWindow >(MaxTxWin/2)<br>**or** With-Invalid-ReqSeq | Close Channel | |
| Recv I-frame | With-Invalid-TxSeq<br>**and** TxWindow ≤ (MaxTx-Win/2) | Close Channel | |
| | | Ignore | RECV |
| Recv RRorRNR | With-Invalid-ReqSeq | Close Channel | |
| Recv REJorSREJ | With-Invalid-ReqSeq-Retrans | Close Channel | |
| Recv frame | | Ignore | RECV |

*Table 8.6: RECV_State table*

### *8.6.5.10 REJ_SENT State Table*

| ***Event*** | **Condition** | **Action** | **Next State** |
|---|---|---|---|
| *Recv I-frame (F=0)* | *With-Expected-TxSeq* **and** *With-Valid-ReqSeq* **and** *With-Valid-F-bit* | *Increment-ExpectedTxSeq*<br>*PassToTx*<br>*Data-Indication*<br>*Send-Ack(F=0)* | RECV |
| *Recv I-frame (F=1)* | *With-Expected-TxSeq* **and** *With-Valid-ReqSeq* **and** *With-Valid-F-bit* | *Increment-ExpectedTxSeq*<br>*PassToTx*<br>*Data-Indication*<br>**If** RejActioned = FALSE **then**<br>*Retransmit I-frames*<br>*Send-Pending-I-frames*<br>**else**<br>RejActioned := FALSE<br>*Send-Ack (F=0)* | RECV |
| *Recv I-frame* | *With-Unexpected-TxSeq* **and** *With-Valid-ReqSeq* **and** *With-Valid-F-bit* | *PassToTx* | REJ_SENT |
| *Recv RR (F=1)* | *With-Valid-ReqSeq* **and** *With-Valid-F-bit* | RemoteBusy := FALSE<br>PassToTx<br>**If** RejActioned = FALSE **then**<br>Retransmit-I-frames<br>**else**<br>RejActioned := FALSE<br>*Send-Pending-I-frames*[ | REJ_SENT |
| *Recv RR(P=0)(F=0)* | *With-Valid-ReqSeq* **and** *With-Valid-F-bit* | *PassToTx*<br>**If** RemoteBusy = TRUE **and** UnackedFrames > 0 **then**<br>Start-RetransTimer<br>RemoteBusy := FALSE<br>*Send-Ack(F=0)* | REJ_SENT |
| *Recv RR(P=1)* | *With-Valid-ReqSeq* **and** *With-Valid-F-bit* | *PassToTx*<br>**If** RemoteBusy = TRUE **and** UnackedFrames > 0 **then**<br>Start-RetransTimer<br>RemoteBusy := FALSE<br>Send RR(F=1) | REJ_SENT |

*Table 8.7: REJ_SENT State table*

| ***Event*** | **Condition** | **Action** | **Next State** |
|---|---|---|---|
| *Recv RNR(P=1)* | *With-Valid-ReqSeq* **and** *With-Valid-F-bit* | RemoteBusy := TRUE<br>*PassToTx*<br>*Send RR(F=1)* | REJ_SENT |
| *Recv RNR(P=0)* | *With-Valid-ReqSeq and With-Valid-F-bit* | RemoteBusy := TRUE<br>*PassToTx*<br>*Send RR(F=0)* | REJ_SENT |
| *Recv REJ (F=0)* | *With-Valid-ReqSeq - Retrans* **and** RetryIframes[i] < MaxTrans-mit **and** *With-Valid-F-bit* | RemoteBusy := FALSE<br>*PassToTx*<br>*Retransmit-I-frames*<br>*Send-Pending-I-frames*<br>**If** *PbitOutstanding* **then**<br>RejActioned := TRUE | REJ_SENT |
| *Recv REJ (F=1)* | *With-Valid-ReqSeq - Retrans* **and** RetryIframes[i] < MaxTrans-mit **and** *With-Valid-F-bit* | RemoteBusy := FALSE<br>*PassToTx*<br>**If** RejActioned = FALSE **then**<br>*Retransmit-I-frames*<br>**else**<br>RejActioned := FALSE<br>Send-Pending-I-frames | REJ_SENT |
| *Recv SREJ (P=0) (F=0)* | *With-Valid-ReqSeq-Retrans* **and** RetryIframes[i] < MaxTrans-mit **and** *With-Valid-F-bit* | RemoteBusy := FALSE<br>*PassToTxFbit*<br>*Retransmit-Requested-I-frame*<br>**If** *PbitOutstanding* **then**<br>SrejActioned := TRUE<br>*SrejSaveReqSeq := ReqSeq* | REJ_SENT |
| *Recv SREJ (P=0) (F=1)* | *With-Valid-ReqSeq-Retrans* **and** RetryIframes[i] < MaxTrans-mit **and** *With-Valid-F-bit* | RemoteBusy := FALSE<br>*PassToTxFbit*<br>**If** SrejActioned = TRUE **and** SrejSaveReqSeq = ReqSeq **then**<br>SrejActioned := FALSE<br>**else**<br>*Retransmit-Requested-I-frame* | REJ_SENT |

*Table 8.7: REJ_SENT State table*

| ***Event*** | **Condition** | **Action** | **Next State** |
|---|---|---|---|
| *Recv SREJ (P=1)* | *With-Valid-ReqSeq-Retrans* **and** RetryIframes[i] < MaxTransmit **and** *With-Valid-F-bit* | RemoteBusy := FALSE<br>*PassToTx*<br>*Retransmit-Requested-I-frames*<br>*Send-Pending-I-frames*<br>**If** *PbitOutstanding* **then** SrejActioned := TRUE<br>*SrejSaveReqSeq := ReqSeq* | REJ_SENT |
| *Recv REJ* | *With-Valid-ReqSeq-Retrans* **and** RetryIframes[i] ≥ MaxTransmit | *Close Channel* | |
| *Recv SREJ (P=0)* | *With-Valid-ReqSeq-Retrans* **and** RetryIframes[i] ≥ MaxTransmit | *Close Channel* | |
| *RECV SREJ(P=1)* | *With-Valid-ReqSeq-Retrans* **and** RetryIframes[i] ≥ MaxTransmit | *Close Channel* | |
| *Recv I-frame* | *(With-Invalid-TxSeq* **and** *TxWindow > (MaxTxWin/2)* **or** *With-Invalid-ReqSeq* | *Close Channel* | |
| *Recv RROrRNR* | *With-Invalid-ReqSeq* | *Close Channel* | |
| *RecvREJorSREJ* | *With-Invalid-ReqSeq-Retrans* | *Close Channel* | |
| *Recv I-frame* | *With-Invalid-TxSeq* | *Close Channel* | |
| | **and** *TxWindow ≤ (MaxTxWin/2)* | *Ignore* | REJ_SENT |
| *Recv frame* | | *Ignore* | REJ_SENT |

*Table 8.7: REJ_SENT State table*

### *8.6.5.11 SREJ_SENT State Table*

| Event | Condition | Action | Next State |
|---|---|---|---|
| Recv I-frame | With-Expected-TxSeq<br>-Srej<br>**and** With-Valid-ReqSeq<br>**and** *With-Valid-F-bit*<br>**and** SendRej = FALSE<br>**and** SrejList = 1 | SaveIframeSrej<br>PopSrejList<br>PassToTx<br>Data-IndicatioSrej<br>BufferSeq := BufferSeqS-rej<br>Send-Ack (F=0) | RECV |
| Recv I-frame | With-Expected-TxSeq-Srej<br>**and** *With-Valid-ReqSeq*<br>**and** *With-Valid-F-bit*<br>**and** SendRej = TRUE<br>**and** SrejList = 1 | SaveIframeSrej<br>PopSrejList<br>PassToTx<br>Data-IndicationSrej<br>BufferSeq := BufferSeqS-rej<br>Send REJ | REJ_SENT |
| Recv I-frame | With-Expected-TxSeq-Srej<br>**and** With-Valid-ReqSeq<br>**and** *With-Valid-F-bit*<br>**and** SrejList > 1 | SaveIframeSrej<br>PopSrejList<br>PassToTx<br>Data-IndicationSrej | SREJ_SENT |
| Recv I-frame | With-Expected-TxSeq<br>**and** With-Valid-ReqSeq<br>**and** *With-Valid-F-bit* | SaveIframe<br>Increment-ExpectedTxSeq<br>PassToTx | SREJ_SENT |
| Recv I-frame | With-Unexpected-TxSeq<br>**and** With-Valid-ReqSeq<br>**and** With-Valid-F-bit<br>**and** SendRej = FALSE | SaveIframeSrej<br>PassToTx<br>Send SREJ | SREJ_SENT |
| | | PassToTx<br>SendRej := TRUE | SREJ_SENT |
| Recv I-frame | With-Unexpected-TxSeq<br>**and** With-Valid-ReqSeq<br>**and** With-Valid-F-bit<br>**and** SendRej = TRUE | PassToTx | SREJ_SENT |
| Recv I-frame | With-Unexpected-TxSeq-Srej<br>**and** With-Valid-ReqSeq<br>**and** With-Valid-F-bit | SaveIframeSrej<br>PassToTx<br>Send SREJ(SrejList) | SREJ_SENT |

*Table 8.8: SREJ_SENT State Table*

| Event | Condition | Action | Next State |
|---|---|---|---|
| Recv I-frame | With-duplicate-TxSeq-Srej<br>**and** With-Valid-ReqSeq<br>**and** With-Valid-F-bit | PassToTx | SREJ_SENT |
| Recv RR(F=1) | With-Valid-ReqSeq<br>**and** With-Valid-F-bit | RemoteBusy := FALSE<br>PassToTx<br>**If** RejActioned = FALSE **then**<br>Retransmit-I-frames<br>else<br>RejActioned := FALSE<br>*Send-Pending-I-frames*] | SREJ_SENT |
| Recv RR(P=1) | With-Valid-ReqSeq<br>**and** With-Valid-F-bit | PassToTx<br>**If** RemoteBusy = TRUE **and** UnackedFrames > 0 **then**<br>Start-RetransTimer<br>RemoteBusy := FALSE<br>Send SREJ(SrejList-tail)(F=1) | SREJ_SENT |
| Recv RR(P=0)(F=0) | With-Valid-ReqSeq<br>**and** With-Valid-F-bit | PassToTx<br>**If** RemoteBusy = TRUE **and** UnackedFrames > 0 **then**<br>Start-RetransTimer<br>RemoteBusy := FALSE<br>Send-Ack(F=0) | SREJ_SENT |
| Recv RNR(P=1) | With-Valid-ReqSeq<br>**and** With-Valid-F-bit | RemoteBusy := TRUE<br>PassToTx<br>Send SREJ(SrejList-tail)(F=1) | SREJ_SENT |
| Recv RNR(P=0) | With-Valid-ReqSeq<br>**and** With-Valid-F-bit | RemoteBusy := TRUE<br>PassToTx<br>Send RR(F=0) | SREJ_SENT |

*Table 8.8: SREJ_SENT State Table*

| Event | Condition | Action | Next State |
|---|---|---|---|
| Recv REJ (F=0) | With-Valid-ReqSeq-Retrans<br>**and**<br>RetryIframes[i] < Max-Transmit<br>**and** *With-Valid-F-bit* | RemoteBusy := FALSE<br>PassToTx<br>Retransmit-I-frames<br>Send-Pending-I-frames<br>**If** *PbitOutstanding* **then**<br>RejActioned := TRUE | SREJ_SENT |
| Recv REJ (F=1) | With-Valid-ReqSeq-Retrans<br>**and**<br>RetryIframes[i] < Max-Transmit<br>**and** *With-Valid-F-bit* | RemoteBusy := FALSE<br>PassToTx<br>**If** RejActioned = FALSE **then**<br>Retransmit-I-frames<br>else<br>RejActioned := FALSE<br>Send-Pending-I-frames | SREJ_SENT |
| Recv SREJ(P=0) (F=0) | With-Valid-ReqSeq-Retrans<br>**and** RetryIframes[i] < MaxTransmit<br>**and** *With-Valid-F-bit* | RemoteBusy := FALSE<br>PassToTxFbit<br>Retransmit-Requested-I-frame<br>**If** *PbitOutstanding* **then**<br>SrejActioned := TRUE<br>*SrejSaveReqSeq* = *ReqSeq* | SREJ_SENT |
| Recv SREJ(P=0) (F=1) | With-Valid-ReqSeq-Retrans<br>**and** RetryIframes[i] < MaxTransmit<br>**and** *With-Valid-F-bit* | RemoteBusy := FALSE<br>PassToTxFbit<br>**If** SrejActioned = TRUE **and**<br>SrejSaveReqSeq = ReqSeq **then**<br>SrejActioned := FALSE<br>else<br>Retransmit-Requested-I-frame | SREJ_SENT |
| Recv SREJ(P=1) | With-Valid-ReqSeq-Retrans<br>**and** RetryIframes[i] < MaxTransmit<br>**and** *With-Valid-F-bit* | RemoteBusy := FALSE<br>PassToTx<br>Retransmit-Requested-I-frame<br>Send-Pending-I-frames<br>**If** *PbitOutstanding* **then**<br>SrejActioned := TRUE<br>*SrejSaveReqSeq* = *ReqSeq* | SREJ_SENT |

*Table 8.8: SREJ_SENT State Table*

| Event | Condition | Action | Next State |
|---|---|---|---|
| Recv REJ | With-Valid-ReqSeq-Retrans **and** RetryIframes[i] ≥ MaxTransmit | Close Channel | |
| Recv SREJ(P=0) | With-Valid-ReqSeqRe-trans **and** RetryIframes[i] ≥ Max-Transmit | Close Channel | |
| Recv SREJ(P=1) | With-Valid-ReqSeqRe-trans **and** RetryIframes[i] ≥ Max-Transmit | Close Channel | |
| Recv I-frame | (*With-Invalid-TxSeq* **and** *TxWindow > (MaxTx-Win/2)* or With-Invalid-ReqSeq | Close Channel | |
| Recv RRor-RNR | With-Invalid-ReqSeq | Close Channel | |
| Recv REJorSREJ | With-Invalid-ReqSeq-Retrans | Close Channel | |
| Recv I-frame | With-Invalid-TxSeq | Close Channel | |
| | **and** TxWindow ≤ (Max-TxWin/2) | Ignore | SREJ_SENT |
| Recv frame | - | Ignore | SREJ_SENT |

*Table 8.8: SREJ_SENT State Table*

## 8.7 STREAMING MODE

When a link is configured to work in Streaming Mode, the frame format for outgoing data is the same as for Enhanced Retransmission mode but frames are not acknowledged. Therefore

- RR, REJ, RNR and SREJ frames shall not be used in Streaming Mode.
- The F-bit shall always be set to zero in the transmitter, and shall be ignored in the receiver.
- the MonitorTimer and RetransmissionTimer shall not be used in Streaming mode.

A channel configured to work in Streaming mode shall be configured with a finite value for the Flush Timeout on the transmitter.

### 8.7.1 Transmitting I-frames

When transmitting a new I-frame the control field parameter ReqSeq shall be set to 0, TxSeq shall be set to NextTXSeq and NextTXSeq shall be incremented by one.

### 8.7.2 Receiving I-frames

Upon receipt of a valid I-frame with TxSeq equal to ExpectedTxSeq, the frame shall be made available to the reassembly function. ExpectedTxSeq shall be incremented by one.

Upon receipt of a valid I-frame with an out-of-sequence TxSeq (see Section 8.7.3.1 on page 185) all frames with a sequence number less than TxSeq shall be assumed lost and marked as missing. The missing I-frames are in the range from ExpectedTxSeq (the frame that the device was expecting to receive) up to and including TxSeq-1. ExpectedTxSeq shall be set to TxSeq +1. The received I-frame shall be made available for pulling by the reassembly function. The ReqSeq shall be ignored.

**Note**: It is possible for a complete window size of I-frames to be missing and thus, no missing I-frames are detected. For example, when a window size of 63 is used this situation occurs when 63 I-frames in a row are missing. If the ability to not detect missing I-frames will cause problems for an application, it is recommended that the Extended Window Size option be used.

If there is no buffer space for the received I-frame an existing I-frame (i.e. the oldest) shall be discarded (flushed) freeing up buffer space for the new I-frame. The discarded I-frame shall be marked as missing.

### 8.7.3 Exception Conditions

Exception conditions may occur as the result of physical layer errors or L2CAP procedural errors. The error recovery procedures which are available following the detection of an exception condition at the L2CAP layer in Streaming mode are defined in this section.

#### *8.7.3.1 TxSeq Sequence error*

A TxSeq sequence error exception condition occurs in the receiver when a valid I-frame is received which contains a TxSeq value which is not equal to the expected value, thus TxSeq is not equal to ExpectedTxSeq.

The out-of-sequence I-frame is identified by a TxSeq that is greater than ExpectedTxSeq (TxSeq > ExpectedTXSeq). The ReqSeq shall be ignored. The missing I-frame(s) are considered lost and ExpectedTXSeq is set equal to TxSeq+1 as specified in Section 8.7.2 on page 185. The missing I-frame(s) are reported as lost to the SDU reassembly function.

# 9 PROCEDURE FOR AMP CHANNEL CREATION AND HANDLING

When an AMP is used, the procedures defined in this chapter shall be used. Enhanced Retransmission mode is used on all reliable channels to ensure a reliable move channel operation and to ensure that user traffic with an infinite flush timeout is reliable even for AMPs that do not support infinite flush timeout. Streaming mode is used on all channels configured with a finite flush timeout.

## 9.1 CREATE CHANNEL

Create Channel Request is used to create a new L2CAP channel over a Controller. The channel will have the quality of service specified by a pair of Extended Flow Specifications exchanged during channel configuration. Channels shall only be created over the BR/EDR controller if both L2CAP entities support Extended Flow Specification for BR/EDR. After configuration each device will have an outgoing (transmit traffic) flow specification and an incoming (received traffic) flow specification. Note: Where Extended Flow Specification for BR/EDR is not supported by one or both L2CAP entities, an L2CAP channel can be established with Connection Request.

All Best Effort channels created over the same AMP Physical link are aggregated over a single AMP Logical link, so if a Best Effort channel is requested over an AMP Physical link where a Best Effort logical link already exists then the Best Effort Extended Flow Specifications are aggregated into one pair of flow specifications and the flow specification of the AMP Logical link is modified using the HCI Flow Spec Modify command (in devices that support HCI). Section 7.8 describes how Best Effort Flow specifications are aggregated. A logical link is created for each Guaranteed channel and one for the first Best Effort channel of the AMP.

All channels created over the same BR/EDR physical link are aggregated over a single logical link. Aggregation of Best Effort Extended Flow Specifications is not necessary for channels created over ACL-U logical links so the term “modify the logical link” in the algorithm descriptions results in “no action” when the logical link is ACL-U. The L2CAP layer should perform admission control for Guaranteed channels created on ACL-U logical links (see Section 7.10 for a description of L2CAP admission control). The term “create a logical link” in the algorithm description refers to L2CAP performing admission control when the logical link is ACL-U.

**Basic Algorithm:**

1. Extended Flow specifications and other configuration parameters shall be exchanged via the Lockstep Configuration procedure.
2. If the service type of the Extended Flow Specifications are Best Effort (including the case where one is Best Effort and the other is “No Traffic”) then the Best Effort Algorithm shall be used otherwise the Guaranteed Algorithm shall be used. (See Section 5.6 for rules on setting the service type of Extended Flow Specifications).

**Best Effort Algorithm:**

1. If one or more Best Effort channels already exist for the Controller then Goto step 3
2. Tell the Controller to create a logical link passing the Extended Flow Specifications. Goto step 5
3. Aggregate the Extended Flow Specifications with all the other Best Effort Extended Flow specifications running on the same physical link as described in Section 7.8.
4. Tell the controller to modify the logical link passing the aggregated Extended Flow Specifications.
5. If the Controller accepts the create/modify request and returns success then complete the L2CAP Configuration with result = success otherwise complete the L2CAP configuration with result = “Failure - flow spec rejected”

**Guaranteed Algorithm**

1. Tell the Controller to create a logical link passing the Extended Flow Specifications.
2. If the Controller accepts the create request and returns success, complete the L2CAP Configuration with result = success; otherwise, complete the L2CAP configuration with result = “Failure - flow spec rejected.”

If the Controller is a BR/EDR or BR/EDR/LE Controller and the Extended Flow Specification type is Guaranteed then L2CAP should perform admission control by determining if the requested QoS can be achieved by the Controller without compromising existing Guaranteed channels running on the Controller.

An example of the creation of the first Best Effort channel or a guaranteed channel is shown in Figure 9.1. An example of creation of a subsequent Best Effort channel is shown in Figure 9.2.

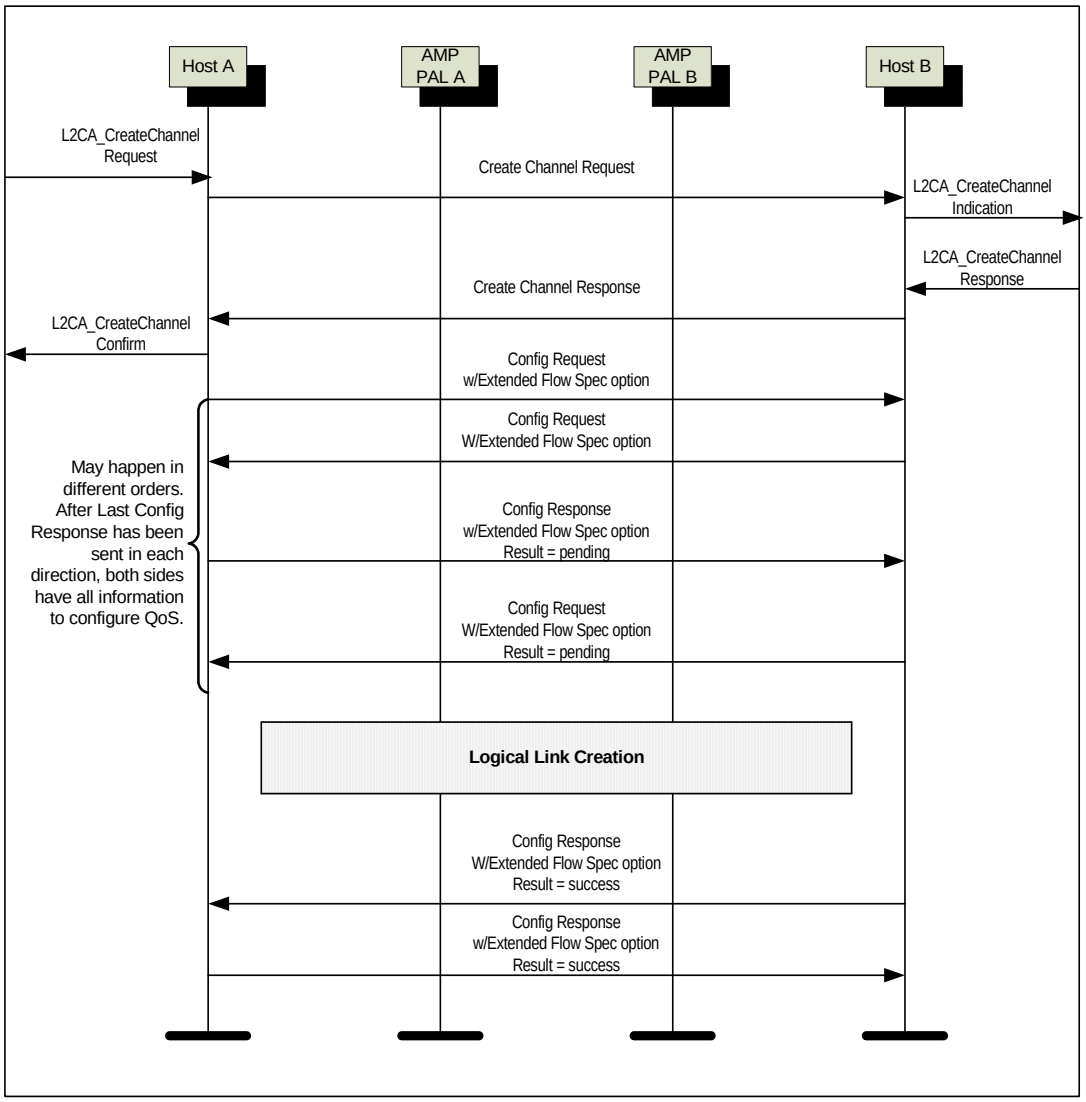



*Figure 9.1: Creation of First Best Effort L2CAP channel or a Guaranteed L2CAP Channel*

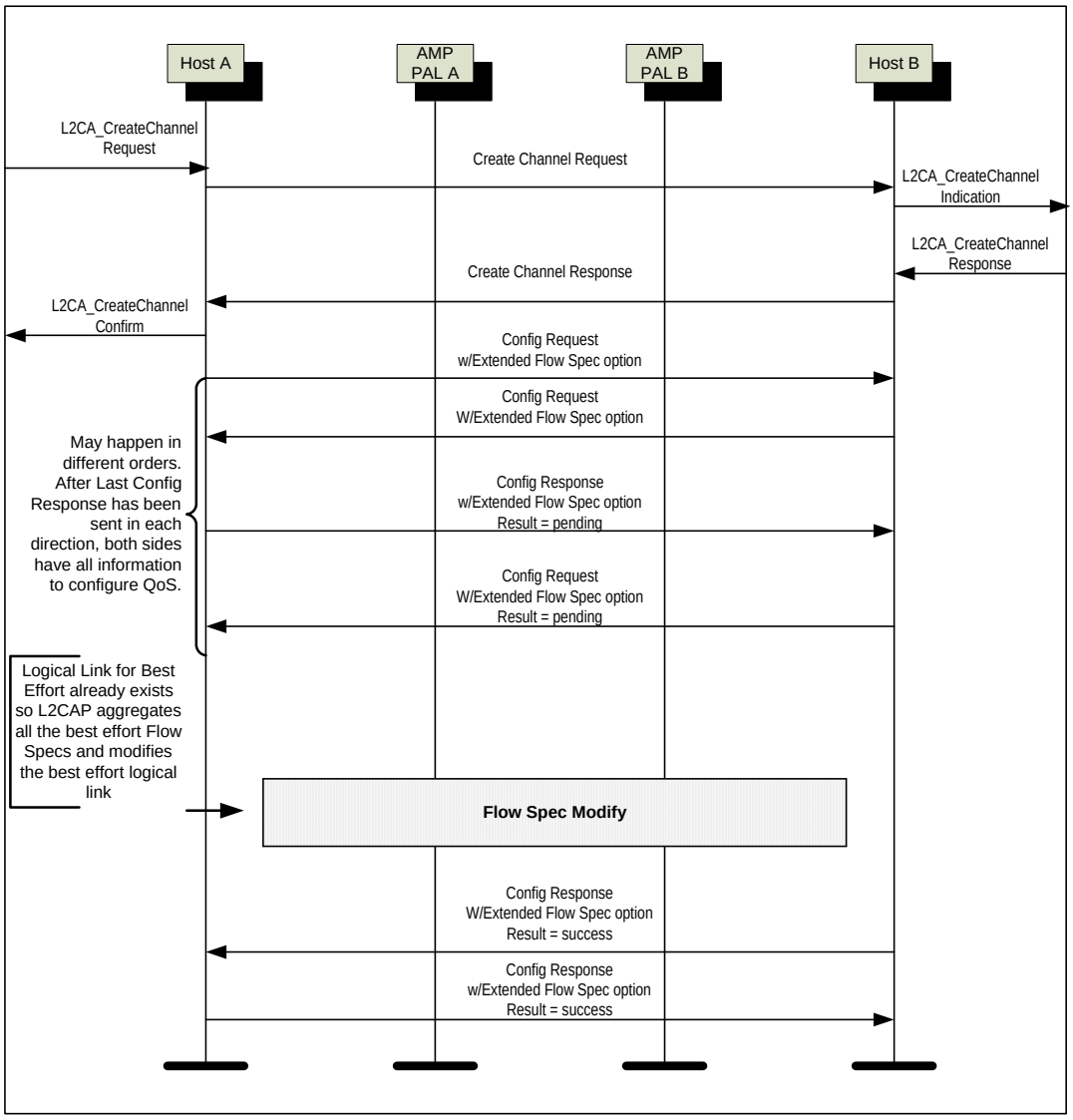



*Figure 9.2: Creation of Subsequent Best Effort L2CAP Channel*

## 9.2 MOVE CHANNEL

Move Channel is used to move an L2CAP channel from one Controller to another. Moving a channel retains the existing channel configuration (MTU, QOS, etc.) and the CID. If reconfiguration is necessary then it may be done after a successful move channel operation. If a channel configured as Best Effort is moved, the Extended Flow Specification aggregate shall be recalculated (see Section 7.8 for the aggregation algorithm) for both the old and new Controllers in cases where an aggregate already exists or where a channel is moved to a Controller with an existing Best Effort channel. If the Extended Flow Specification aggregate are modified for a Controller then a logical link modify operation shall be performed for that Controller.

The two procedures for moving channels are based on the mode configured for the channel. One procedure is used for channels configured with Enhanced

Retransmission mode. Another procedure is used for channels configured with Streaming mode. The procedures are described in the following sections.

Note that the terms such as “logical link create” and “logical link modify” are used in the Move procedures. For AMP Controllers these terms refer to logical link operations (e.g. QoS admission control) performed by the Controller. For BR/EDR and BR/EDR/LE Controllers these terms refer to QoS admission control performed by L2CAP on behalf of the Controller.

### 9.2.1 Move Channel Protocol Procedure with Enhanced Retransmission Mode

1. When the L2CAP layer on the initiating device receives the L2CA_Move_Channel.request from the upper layer (application) it shall stop sending I-frames and S-frames. It shall continue to listen on the old Controller and follow the procedures in Section 9.2.1.1. The L2CAP layer on the initiating device shall send the Move Channel Request packet over the L2CAP signaling channel to the remote (responding) device.
2. When the L2CAP layer on the responding device receives the Move Channel Request packet it shall stop sending I-frames and S-frames. It shall still listen on the old Controller and follow the procedures in Section 9.2.1.1. It shall send a Move Channel Response packet to the other side. If a logical link must be created or modified (or admission control is required) the Move Channel Response packet shall contain a result code of “pending.” If a logical link does not need to be created or modified and the move operation is allowed then the Move Channel Response packet shall contain a result code of “success.” Otherwise, it shall contain the appropriate “refused” result code. If the response code sent in the Move Channel Response packet is “pending” then the L2CAP layer on the responding device shall attempt to create or modify a logical link on the new Controller for the channel. When the logical link create/modify operation is complete, the L2CAP layer on the responding device shall send a Move Channel Response packet to the other side with the result code indicating the success/failure of the logical link create/modify.
3. When the L2CAP layer on the initiating device receives the Move Channel Response packet it shall check the result code. If the result code in the Move Channel Response packet is “refused” it shall send a Move Channel Confirmation packet with result code “failure” to the other side. If the result code in the Move Channel Response packet is “pending” or “success” the L2CAP layer shall stop listening on the old Controller and attempt to create/modify a logical link on the new Controller for the channel. If the result code in the Move Channel Response packet is “pending” then it shall use the ERTX timer while waiting for a Move Channel Response packet with a “non-pending” result code from the responding device. When the logical link create/modify operation is complete and the L2CAP layer on the initiating

device has received a Move Channel Response packet from the responding device with a “non-pending” result code, it shall send a Move Channel Confirmation packet to conclude the Move Channel procedure. The result code in the Move Channel Confirmation packet indicates the success/failure of its own logical channel create/modify operation and the result code in the Move Channel Response packet from the responding device. If both are success then the result code sent in the Move Channel Confirmation packet shall be “success” otherwise it shall be “failure.”

4. When the L2CAP layer on the responding device receives the Move Channel Confirmation packet it shall send a Move Channel Confirmation Response and send an L2CA_Move_Channel_Confirm.indication to the upper layer with the result code contained in the Move Channel Confirmation packet. If the result code of the Move Channel Confirmation is success the L2CAP layer on the responding device shall listen on the new Controller if the result code is failure then it shall listen on the old Controller.
5. When the L2CAP layer on the initiating device receives the Move Channel Confirmation response packet it shall send an L2CAP_Move_Channel.confirm to the upper layer with the result code passed in the Move Channel Confirmation packet. If the result code sent in the Move Channel Confirmation packet is success then the L2CAP layer on the initiating device shall send an RR(P=1) on the new Controller and shall start listening on the new Controller. Before sending the RR(P=1) the L2CAP layer on the initiating device may initiate reconfiguration by sending a Configuration Request packet. If the result sent is failure then the L2CAP layer on the initiating device shall send an RR(P=1) on the old Controller and start listening on the old Controller.
6. When the L2CAP layer on the responding device receives the RR(P=1) packet it shall send an I-frame(F=1) with a TxSeq matching the ReqSeq in the RR(P=1) or an RR(F=1) if there are no I-frames to send. If the I-frame is a retransmission and the PDU size used for the new Controller is smaller than the PDU size used for the old Controller, the responding device shall segment the retransmitted I-frame to make it fit in the new PDU size. If the L2CAP layer on the responding device receives a Configuration Request packet before the RR(P=1) it shall perform reconfiguration. When the reconfiguration is complete it shall wait for the RR(P=1). If the L2CAP layer on the responding device desires to perform reconfiguration but receives the RR(P=1) it may send a Configuration Request packet instead of the RR(F=1) or I-frame(F=1) to initiate reconfiguration. After the reconfiguration is complete it shall send the RR(F=1) or I-frame(F=1).

7. When the L2CAP layer on the initiating device receives the I-frame(F=1) or RR(F=1) from the responding device, the move is complete. If it receives a Configuration Request packet before receiving the RR(F=1) or I-frame(F=1) it shall perform reconfiguration. When reconfiguration is complete it shall wait for the RR(F=1) or I-frame(F=1).

#### *9.2.1.1 Enhanced Retransmission Mode Procedures During A Move Operation*

1. Stop sending I-frames and S-frames and cancel all timers. Set the transmitter to the XMIT state and clear all retry counters.
2. Clear or reset any SREJ_SENT and REJ_SENT state specific variables including flushing any received out-of-sequence I-frames saved as part of being in the SREJ_SENT state. Set the receiver to the RECV state.
3. Process received I-frames in the RECV state including processing the ReqSeq and passing completed SDUs to the upper layer. Only process the ReqSeq of received S-frames. Do not change state.
4. Ignore all out-of-sequence I-frames keeping note of the TxSeq of the last in-sequence I-frame received.
5. If the initiating device is in a local busy condition then it should wait to send the Move Channel Confirmation packet until the local busy condition has cleared. If the responding device is in a local busy condition then it should send a Move Channel Response packet with result “pending” upon receiving the Move Channel Request packet and wait until the local busy condition is cleared before sending a Move Channel Response packet with a “non-pending” result.
6. The ReqSeq of the RR(P=1) sent by the initiating device shall be set to the TxSeq of the next I-frame after the last in-sequence I-frame received (noted in step 4). The ReqSeq of the RR(F=1) or I-frame(F=1) sent by the responding device shall be set to the TxSeq of the next frame after the last in-sequence I-frame received (noted in step 4).
7. The new Controller or HCI transport may require smaller PDU size than the old Controller. If this is the case then the L2CAP layer shall segment all retransmitted I-frames to fit the new PDU size.
8. After sending the RR(P=1) the initiating device should start the Monitor Timer and go to the WAIT_F state.
9. Timers shall be stopped during channel reconfiguration and restarted when reconfiguration is complete.

### 9.2.2 Move Channel Protocol Procedure with Streaming Mode (Initiator is Data Source)

1. When the L2CAP layer on the initiating device (data source) receives the L2CA_Move_Channel.request it shall stop sending L2CAP PDUs. The L2CAP layer on the initiating device shall send a Move Channel Request packet over the L2CAP signaling channel to the remote (responding) device.
2. When the L2CAP layer on the responding device receives the Move Channel Request packet it shall still listen on the old Controller for receive frames (timing could be such that packets from the initiating device are still in transit). It shall pass completely received SDUs up to the upper layer. It shall send a Move Channel Response packet to the other side. If a logical link must be created (or admission control is required) the Move Channel Response packet shall contain a result code of “pending.” If a logical link does not need to be created and the move operation is allowed then the Move Channel Response packet shall contain a result code of “success.” Otherwise it shall contain the appropriate “refused” result code. If the response code sent in the Move Channel Response packet is “pending” then the L2CAP layer on the responding device shall attempt to create a logical link on the new Controller for the channel. When the logical link create operation is complete, the L2CAP layer on the responding device shall send a Move Channel Response packet to the other side with the result code indicating the success/failure of the logical link create operation. It shall continue to listen on the old Controller.
3. When the L2CAP layer on the initiating device receives the Move Channel Response packet it shall check the result code. If the result code in the Move Channel Response packet is “refused” it shall send a Move Channel Confirmation packet with result code “failure” to the other side. If the result code in the Move Channel Response packet is “pending” or “success” the L2CAP layer shall attempt to create a logical link on the new Controller for the channel. If the result code in the Move Channel Response packet is “pending” then it shall use the ERTX timer while waiting for a Move Channel Response packet with a “non-pending” result code from the responding device. When the logical link create operation is complete and it has received a Move Channel Response packet from the responding device with a “non-pending” result code it shall send a Move Channel Confirmation packet and wait for a Move Channel Confirmation response packet. The result code in the Move Channel Confirmation indicates the success/failure of its own logical link create operation and the result code in the Move Channel Response packet from the responding device. If both are success then the result code sent in the Move Channel Confirmation shall be success otherwise it shall be failure.

4. When the L2CAP layer on the responding device receives the Move Channel Confirmation packet it shall send a Move Channel Confirmation response packet and send an L2CA_Move_Channel_Confirm.indication to the upper layer with the result code passed in the Move Channel Confirmation packet. If the result code of the Move Channel Confirmation is success it shall listen on the new Controller. If the result code is failure it shall listen on the old Controller.
5. When the L2CAP layer on the initiating device receives the Move Channel Confirmation response packet it shall send an L2CAP_Move_Channel.confirm to the upper layer with the result code passed in the Move Channel Confirmation packet. If the result sent in the Move Channel Confirmation is success then the initiating device shall send I-frames for the channel on the new Controller otherwise it shall send I-frames for the channel on the old Controller.

### 9.2.3 Move Channel Protocol Procedure with Streaming Mode (Initiator is Data Sink)

1. When the L2CAP layer on initiating device (data sink) receives the L2CAP_Move_Channel.request it shall send a Move Channel Request packet over the L2CAP signaling channel to the remote (responding) device. It shall continue listening on the old Controller.
2. When the L2CAP layer on the responding device receives the Move Channel Request packet it shall stop sending L2CAP PDUs and send a Move Channel Response packet to the other side. If a logical link must be created (or admission control is required) the Move Channel Response packet shall contain a result code of “pending.” If a logical link does not need to be created and the move operation is allowed then the Move Channel Response packet shall contain a result code of “success.” Otherwise it shall contain the appropriate “refused” result code. If the result code sent in the Move Channel Response packet is “pending” then the L2CAP layer on the responding device shall attempt to create a logical link on the new Controller for the channel. When the logical link create operation is complete, the L2CAP layer on the responding device shall send a Move Channel Response packet to the other side with the result code indicating the success/failure of the logical link create.
3. When the L2CAP layer on the initiating device receives the Move Channel Response packet it shall check the result code. If the result code in the Move Channel Response packet is “refused” it shall send a Move Channel Confirmation packet with result code “failure” to the other side. If the result code in the Move Channel Response packet is “pending” or “success” the L2CAP layer shall attempt to create a logical link on the new Controller for the channel. If the result code in the Move Channel Response packet is “pending” then it shall use the ERTX timer while waiting for a Move Channel Response packet with

a “non-pending” result code from the responding device. When the logical link create operation is complete and it has received a Move Channel Response packet from the responding device with a “non-pending” result code it shall send a Move Channel Confirmation packet and wait for a Move Channel Confirmation response packet. The result code in the Move Channel Confirmation indicates the success/failure of its own logical link create operation and the result code in the Move Channel Response packet from the responding device. If both are success then the result code sent in the Move Channel Confirmation shall be success otherwise it shall be failure. If the result sent is success then the initiating device shall listen on the new Controller otherwise it shall listen on the old Controller.

4. When the L2CAP layer on the responding device receives the Move Channel Confirmation packet it shall send a Move Channel Confirmation response packet and send an L2CA_Move_Channel_Confirm.indication to the upper layer. If the result code in the Move Channel Confirmation packet is success it shall send I-frames for the channel on the new Controller. If the result code is failure it shall send I-frames on for the channel the old Controller.
5. When the L2CAP layer on the initiating device receives the Move Channel Confirmation response packet it shall send an L2CA_Move_Channel.confirm to the upper layer with the result code passed in the Move Channel Confirmation packet.

Figure 9.3 and Figure 9.4 show examples of the move operation with Enhanced Retransmission mode active.

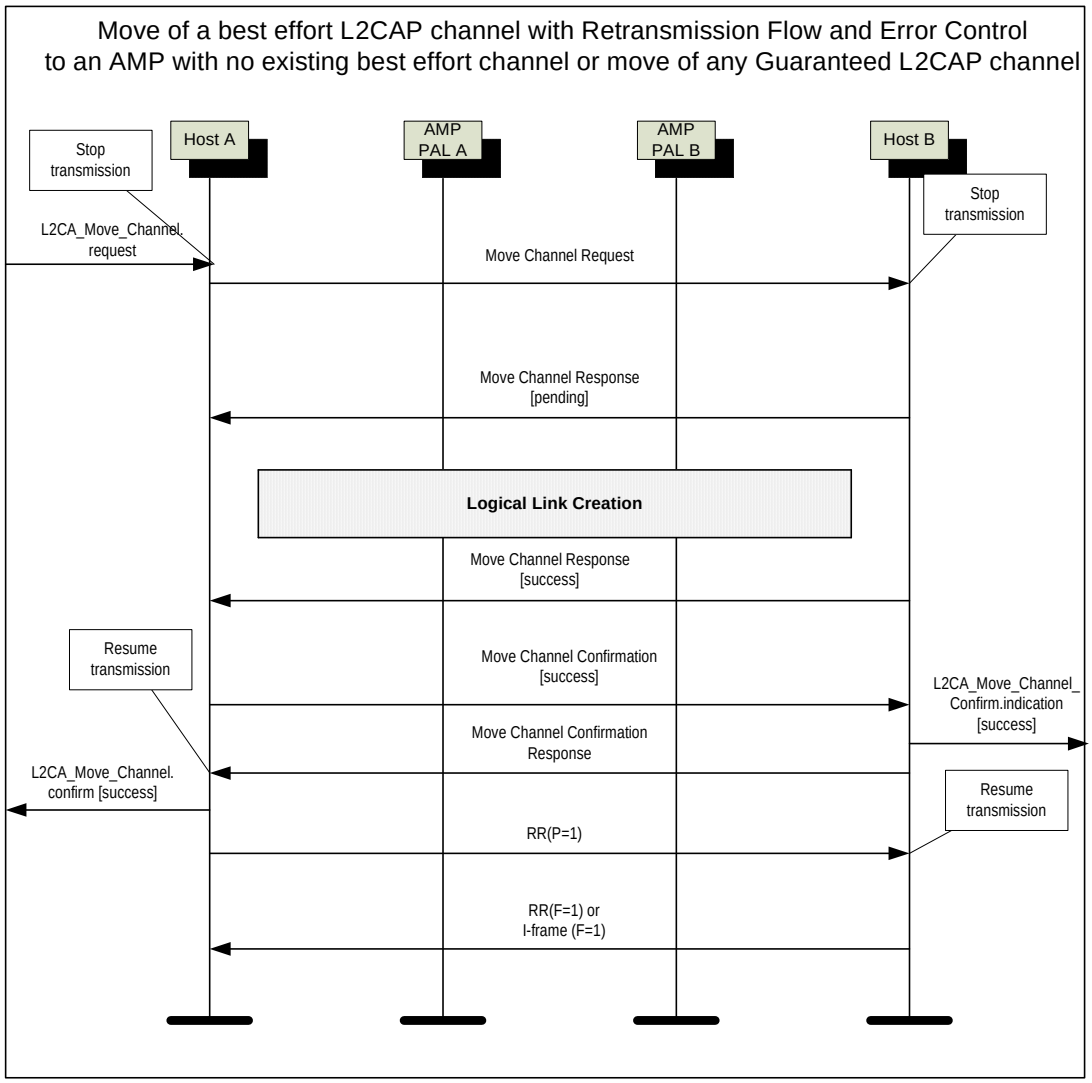



*Figure 9.3: Move of a Best Effort L2CAP channel Enhanced Retransmission mode enabled to an AMP with no existing best effort logical link or move of any Guaranteed L2CAP channel*

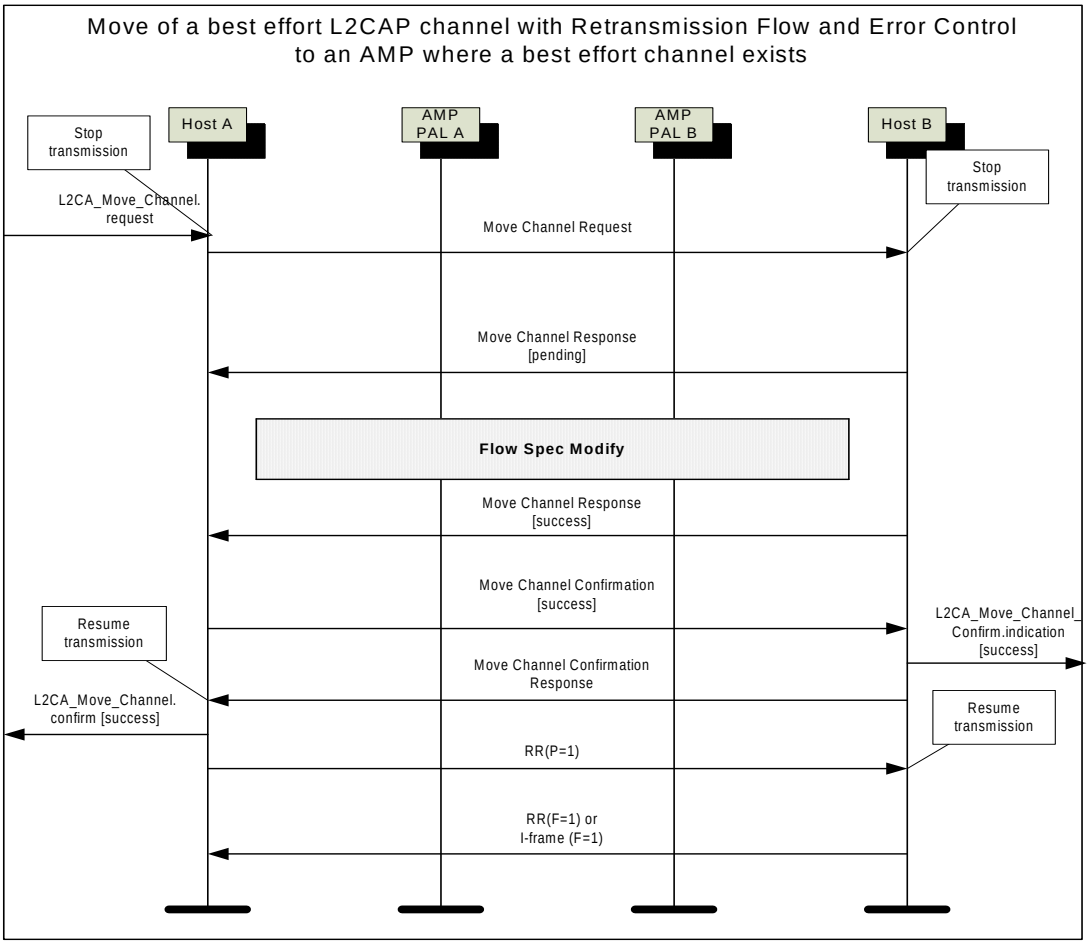



*Figure 9.4: Move of a Best Effort L2CAP channel with Enhanced Retransmission mode enabled to an AMP where a best effort logical link exists*

## 9.3 DISCONNECT CHANNEL

If a channel configured as Best Effort is disconnected the Extended Flow Specification aggregate shall be recalculated (see Section 7.8 for the aggregation algorithm) unless it is the last or only channel. If this is not the last or only Best Effort channel for the logical link then a logical link modify operation shall be performed otherwise the logical link should be disconnected. The ACL-U logical link between two devices should not be disconnected until all the L2CAP channels running over all logical links between the two devices have been disconnected.

# 10 LIST OF FIGURES

Figure 1.1: L2CAP architectural blocks ..... 31
Figure 2.1: Channels between devices ..... 40
Figure 2.2: L2CAP transaction model ..... 41
Figure 3.1: L2CAP PDU format in Basic L2CAP mode on connection-oriented channels (field sizes in bits) ..... 44
Figure 3.2: L2CAP PDU format on the Connectionless channel ..... 45
Figure 3.3: L2CAP PDU formats in Flow Control and Retransmission Modes ..... 46
Figure 3.4: The LFSR circuit generating the FCS. ..... 51
Figure 3.5: Initial state of the FCS generating circuit. ..... 51
Figure 4.1: L2CAP PDU format on a signaling channel ..... 54
Figure 4.2: Command format ..... 55
Figure 4.3: Command Reject packet ..... 57
Figure 4.4: Connection Request Packet ..... 58
Figure 4.5: Connection Response Packet ..... 59
Figure 4.6: Configuration Request Packet ..... 61
Figure 4.7: Configuration Request Flags field format ..... 61
Figure 4.8: Configuration Response Packet ..... 63
Figure 4.9: Configuration Response Flags field format ..... 63
Figure 4.10: Disconnection Request Packet ..... 65
Figure 4.11: Disconnection Response Packet ..... 65
Figure 4.12: Echo Request Packet ..... 66
Figure 4.13: Echo Response Packet ..... 66
Figure 4.14: Information Request Packet ..... 67
Figure 4.15: Information Response Packet ..... 68
Figure 4.16: Create Channel Request Packet ..... 71
Figure 4.17: Create Channel Response Packet ..... 72
Figure 4.18: Move Channel Request Packet ..... 73
Figure 4.19: Move Channel Response Packet ..... 74
Figure 4.20: Move Channel Confirmation Packet ..... 75
Figure 4.21: Move Channel Confirmation Response Packet ..... 76
Figure 4.22: Connection Parameters Update Request packet ..... 77
Figure 4.23: Connection Parameters Update Response packet ..... 78
Figure 5.1: Configuration option format ..... 79
Figure 5.2: MTU Option Format ..... 81
Figure 5.3: Flush Timeout option format. ..... 81
Figure 5.4: Quality of Service (QoS) option format containing Flow Specification ..... 83
Figure 5.5: Retransmission and Flow Control option format. ..... 86
Figure 5.6: FCS option format ..... 91
Figure 5.7: Extended Flow Specification option format ..... 92

Figure 5.8: Extended Window Size option format ....96
Figure 5.9: Extended Window Size values ....97
Figure 6.1: States and transitions. ....117
Figure 6.2: Configuration states and transitions. ....118
Figure 6.3: States and transitions–AMP enabled operation ....119
Figure 7.1: L2CAP fragmentation in a BR/EDR Controller ....127
Figure 7.2: Example of fragmentation processes in a device with a BR/EDR Controller and USB HCI transport. ....128
Figure 7.3: Segmentation and fragmentation of an SDU in a BR/EDR Controller. ....129
Figure 7.4: Example of segmentation and fragment processes in a device with a BR/EDR Controller and USB HCI Transport ....131
Figure 8.1: Example of the transmitter side ....142
Figure 8.2: Example of the receiver side ....143
Figure 8.3: Overview of the receiver side when operating in flow control mode ....151
Figure 8.4: Receiver state machine ....161
Figure 9.1: Creation of First Best Effort L2CAP channel or a Guaranteed L2CAP Channel 1 ....88
Figure 9.2: Creation of Subsequent Best Effort L2CAP Channel ....189
Figure 9.3: Move of a Best Effort L2CAP channel Enhanced Retransmission mode enabled to an AMP with no existing best effort logical link or move of any Guaranteed L2CAP channel ....196
Figure 9.4: Move of a Best Effort L2CAP channel with Enhanced Retransmission mode enabled to an AMP where a best effort logical link exists ....197

# 11 LIST OF TABLES

Table 1.1: Terminology ..... 34
Table 2.1: CID name space ..... 39
Table 2.2: Types of Channel Identifiers ..... 40
Table 3.1: Standard Control Field formats ..... 48
Table 3.3: Extended Control Field formats ..... 49
Table 3.2: Enhanced Control Field formats ..... 49
Table 3.4: SAR control element format. ..... 50
Table 3.5: S control element format: type of S-frame. ..... 50
Table 4.1: Minimum Signaling MTU ..... 54
Table 4.2: Signaling Command Codes ..... 55
Table 4.3: Reason Code Descriptions ..... 57
Table 4.4: Reason Data values ..... 57
Table 4.5: Result values ..... 59
Table 4.6: Status values ..... 60
Table 4.7: Configuration Response Result codes ..... 63
Table 4.8: InfoType definitions ..... 67
Table 4.9: Information Response Result values ..... 68
Table 4.10: Information Response Data fields ..... 69
Table 4.11: Extended feature mask. ..... 69
Table 4.12: Fixed Channels Supported bit mask ..... 70
Table 4.13: Result values ..... 72
Table 4.14: Status values ..... 73
Table 4.15: Result values ..... 74
Table 4.16: Result values ..... 75
Table 5.1: Service type definitions ..... 83
Table 5.2: Mode definitions. ..... 86
Table 5.3: FCS types ..... 92
Table 5.4: Traffic parameters for L2CAP QoS configuration ..... 93
Table 5.5: Traffic parameters for L2CAP QoS configuration ..... 94
Table 6.1: CLOSED state event table. ..... 100
Table 6.2: WAIT_CONNECT_RSP state event table ..... 101
Table 6.3: WAIT_CONNECT state event table. ..... 102
Table 6.4: CONFIG state event table ..... 104
Table 6.5: CONFIG state/substates event table: non-success transitions within configuration process. ..... 105
Table 6.6: CONFIG state/substates event table: events not related to configuration process. ..... 106
Table 6.7: OPEN state event table. ..... 108
Table 6.8: WAIT_DISCONNECT state event table. ..... 109
Table 6.9: WAIT_CREATE_RSP state event table. ..... 110
Table 6.10: WAIT_CREATE state event table ..... 110

Table 6.11: WAIT_MOVE_RSP state event table ..... 111
Table 6.12: WAIT_MOVE state event table ..... 113
Table 6.13: WAIT_MOVE_CONFIRM state event table ..... 114
Table 6.14: WAIT_CONFIRM_RSP state event table ..... 114
Table 7.1: Parameters allowed in Request ..... 121
Table 7.2: Parameters allowed in Response ..... 122
Table 7.3: Permitted parameter in L2CAP Configuration Response for Service Type “Best Effort” ..... 124
Table 7.4: Permitted parameter changes in L2CAP Configuration Response for Service Type “Guaranteed” ..... 124
Table 8.1: Summary of actions when the RetransmissionTimer is in use and R=0. ..... 145
Table 8.2: AMP Controller timeout values ..... 159
Table 8.3: Enhanced Retransmission mode uses the following variables and sequence numbers described in Section 8.3 ..... 162
Table 8.4: XMIT state table ..... 173
Table 8.5: WAIT_F State Table ..... 173
Table 8.6: RECV_State table ..... 175
Table 8.7: REJ_SENT State table ..... 178
Table 8.8: SREJ_SENT State Table ..... 181

# APPENDIX A: CONFIGURATION MSCs

The examples in this appendix describe a sample of the multiple possible configuration scenarios that might occur.

Figure I illustrates the basic configuration process. In this example, the devices exchange MTU information. All other values are assumed to be default.

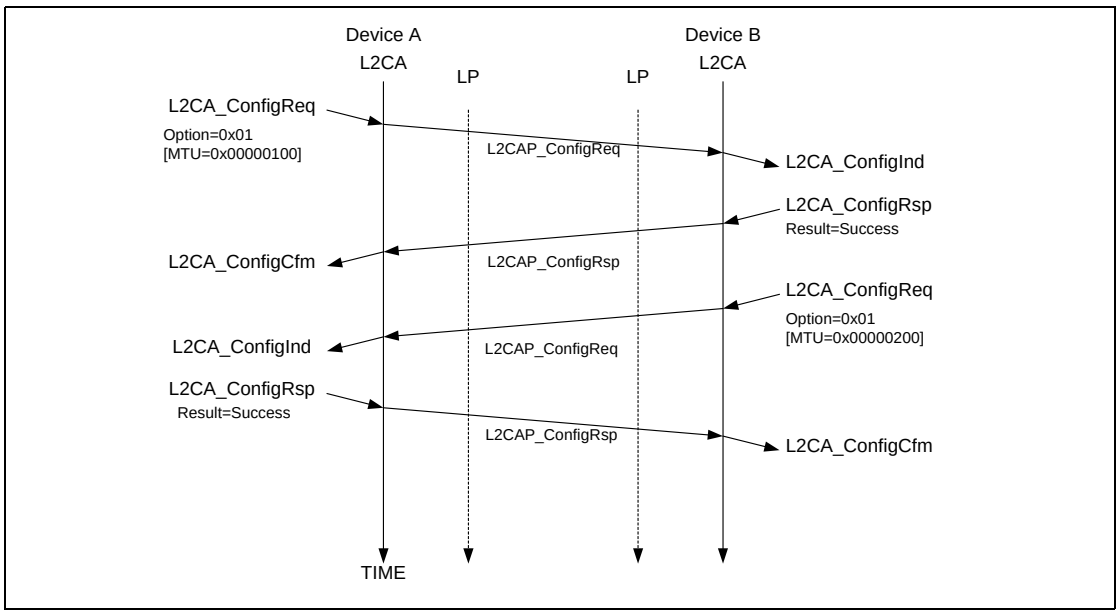



*Figure I: Basic MTU exchange*

Figure II on page 203 illustrates how two devices interoperate even though one device supports more options than the other does. Device A is an upgraded version. It uses a hypothetically defined option type 0x20 for link-level security. Device B rejects the command using the Configuration Response packet with result ‘unknown parameter’ informing Device A that option 0x20 is not understood. Device A then resends the request omitting option 0x20. Device B notices that it does not need to such a large MTU and accepts the request but includes in the response the MTU option informing Device A that Device B will not send an L2CAP packet with a payload larger than 0x80 octets over this channel. On receipt of the response, Device A could reduce the buffer allocated to hold incoming traffic.

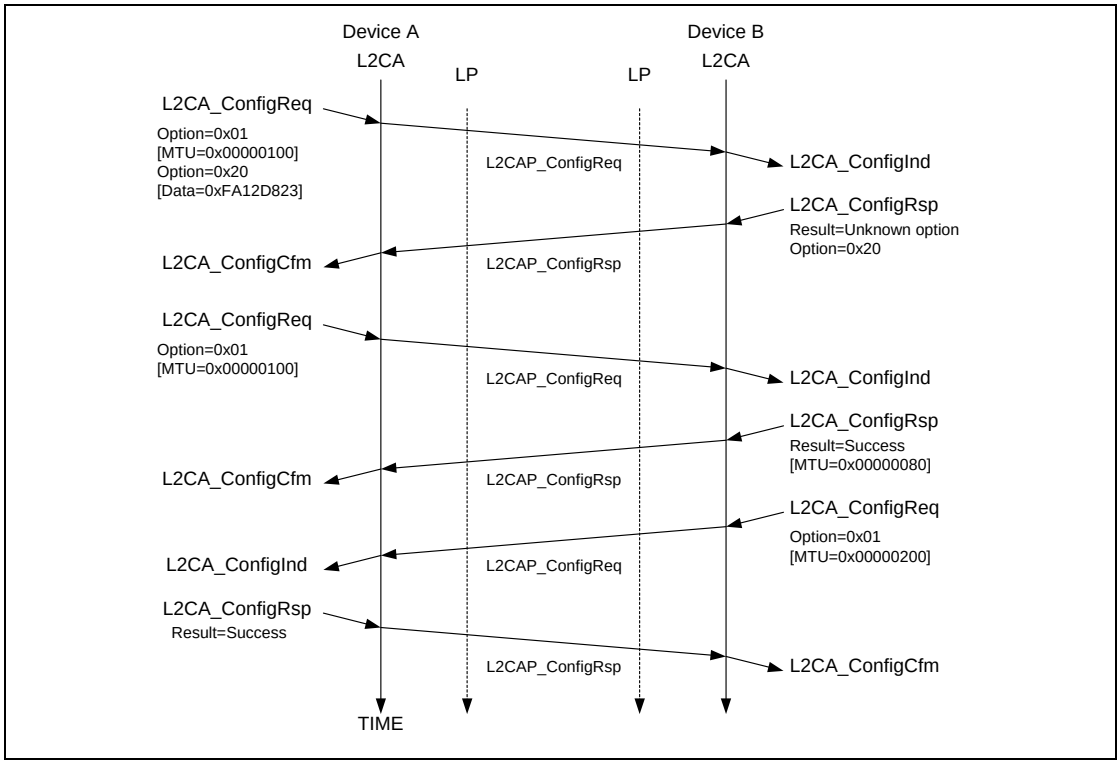



*Figure II: Dealing with Unknown Options*

Figure III on page 204 illustrates an unsuccessful configuration request. There are two problems described by this example. The first problem is that the configuration request is placed in an L2CAP packet that cannot be accepted by the remote device, due to its size. The remote device informs the sender of this problem using the Command Reject message. Device A then resends the configuration options using two smaller L2CAP_ConfigReq messages.

The second problem is an attempt to configure a channel with an invalid CID. For example device B may not have an open connection on that CID (0x01234567 in this example case).

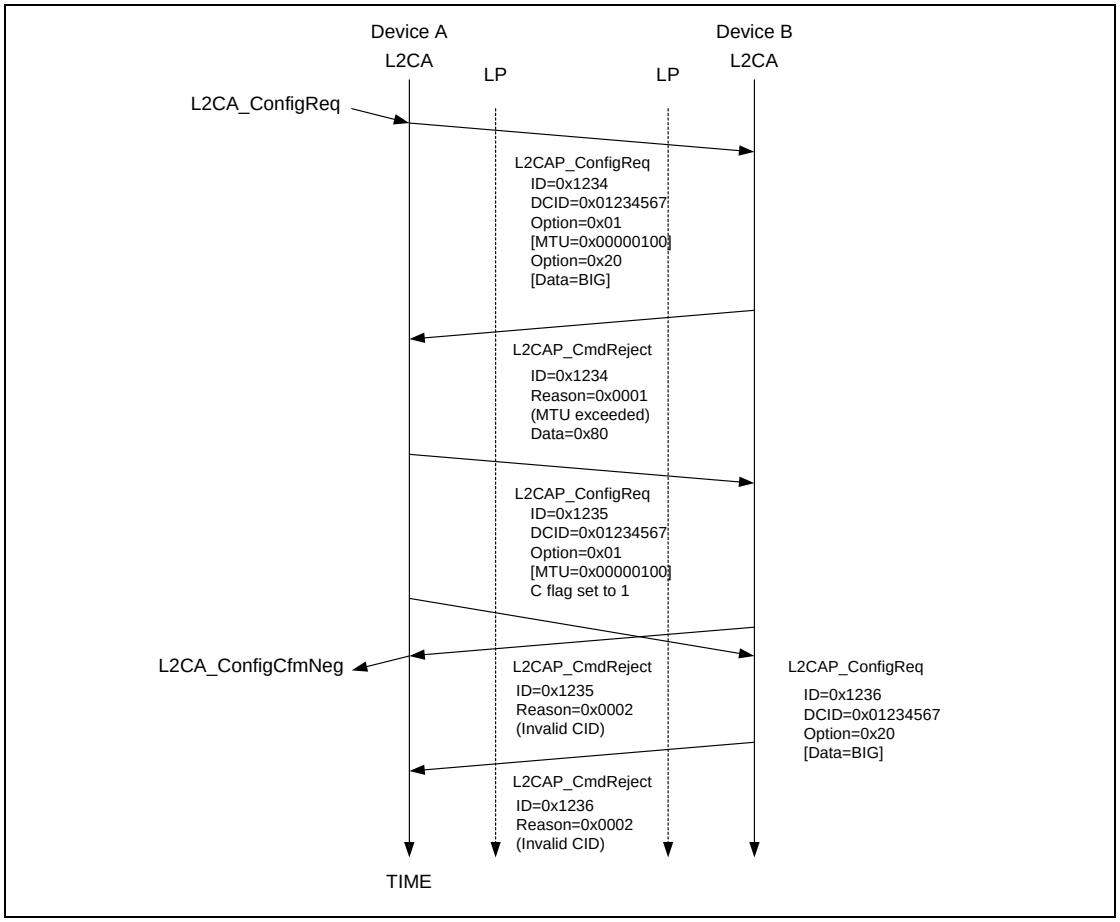



*Figure III: Unsuccessful Configuration Request*

# SERVICE DISCOVERY PROTOCOL (SDP) SPECIFICATION

***This specification defines a protocol for locating services provided by or available through a Bluetooth device.***


Ford Ex. 1052

# CONTENTS

**1 Introduction ..... 209**
1.1 General Description ..... 209
1.2 Motivation ..... 209
1.3 Requirements ..... 209
1.4 Non-requirements and Deferred Requirements ..... 210
1.5 Conventions ..... 210
1.5.1 Bit And Byte Ordering Conventions ..... 210
**2 Overview ..... 211**
2.1 SDP Client-Server Interaction ..... 211
2.2 Service Record ..... 212
2.3 Service Attribute ..... 213
2.3.1 Attribute ID ..... 214
2.3.2 Attribute Value ..... 215
2.4 Service Class ..... 215
2.4.1 A Printer Service Class Example ..... 215
2.5 Searching for Services ..... 216
2.5.1 UUID ..... 216
2.5.2 Service Search Patterns ..... 217
2.6 Browsing for Services ..... 217
2.6.1 Example Service Browsing Hierarchy ..... 218
**3 Data Representation ..... 220**
3.1 Data Element ..... 220
3.2 Data Element Type Descriptor ..... 220
3.3 Data Element Size Descriptor ..... 221
3.4 Data Element Examples ..... 222
**4 Protocol Description ..... 223**
4.1 Transfer Byte Order ..... 223
4.2 Protocol Data Unit Format ..... 223
4.3 Partial Responses and Continuation State ..... 225
4.4 Error Handling ..... 225
4.4.1 SDP_ErrorResponse PDU ..... 226
4.5 ServiceSearch Transaction ..... 227
4.5.1 SDP_ServiceSearchRequest PDU ..... 227
4.5.2 SDP_ServiceSearchResponse PDU ..... 228
4.6 ServiceAttribute Transaction ..... 230
4.6.1 SDP_ServiceAttributeRequest PDU ..... 230
4.6.2 SDP_ServiceAttributeResponse PDU ..... 232
4.7 ServiceSearchAttribute Transaction ..... 233

4.7.1 SDP_ServiceSearchAttributeRequest PDU ..... 233
4.7.2 SDP_ServiceSearchAttributeResponse PDU ..... 235

**5 Service Attribute Definitions ..... 237**
5.1 Universal Attribute Definitions ..... 237
5.1.1 ServiceRecordHandle Attribute ..... 237
5.1.2 ServiceClassIDList Attribute ..... 237
5.1.3 ServiceRecordState Attribute ..... 238
5.1.4 ServiceID Attribute ..... 238
5.1.5 ProtocolDescriptorList Attribute ..... 239
5.1.6 AdditionalProtocolDescriptorList Attribute ..... 240
5.1.7 BrowseGroupList Attribute ..... 241
5.1.8 LanguageBaseAttributeIDList Attribute ..... 241
5.1.9 ServiceInfoTimeToLive Attribute ..... 242
5.1.10 ServiceAvailability Attribute ..... 243
5.1.11 BluetoothProfileDescriptorList Attribute ..... 243
5.1.12 DocumentationURL Attribute ..... 244
5.1.13 ClientExecutableURL Attribute ..... 244
5.1.14 IconURL Attribute ..... 245
5.1.15 ServiceName Attribute ..... 245
5.1.16 ServiceDescription Attribute ..... 246
5.1.17 ProviderName Attribute ..... 246
5.1.18 Reserved Universal Attribute IDs ..... 246
5.2 ServiceDiscoveryServer Service Class Attribute Definitions ... 246
5.2.1 ServiceRecordHandle Attribute ..... 246
5.2.2 ServiceClassIDList Attribute ..... 247
5.2.3 VersionNumberList Attribute ..... 247
5.2.4 ServiceDatabaseState Attribute ..... 247
5.2.5 Reserved Attribute IDs ..... 248
5.3 BrowseGroupDescriptor Service Class Attribute Definitions ... 248
5.3.1 ServiceClassIDList Attribute ..... 248
5.3.2 GroupID Attribute ..... 249
5.3.3 Reserved Attribute IDs ..... 249

**6 Security ..... 250**
**List of Figures ..... 251**
**List of Tables ..... 251**
**Appendix A – Background Information ..... 252**
**Appendix B – Example SDP Transactions ..... 253**

# 1 INTRODUCTION

## 1.1 GENERAL DESCRIPTION

The service discovery protocol (SDP) provides a means for applications to discover which services are available and to determine the characteristics of those available services.

## 1.2 MOTIVATION

Service Discovery in the Bluetooth environment, where the set of services that are available changes dynamically based on the RF proximity of devices in motion, is qualitatively different from service discovery in traditional network-based environments. The service discovery protocol defined in this specification is intended to address the unique characteristics of the Bluetooth environment. See section , “Appendix A – Background Information,” on page 252, for further information on this topic.

## 1.3 REQUIREMENTS

The following capabilities have been identified as requirements for version 1.0 of the Service Discovery Protocol.

1. SDP shall provide the ability for clients to search for needed services based on specific attributes of those services.
2. SDP shall permit services to be discovered based on the class of service.
3. SDP shall enable browsing of services without a priori knowledge of the specific characteristics of those services.
4. SDP shall provide the means for the discovery of new services that become available when devices enter RF proximity with a client device as well as when a new service is made available on a device that is in RF proximity with the client device.
5. SDP shall provide a mechanism for determining when a service becomes unavailable when devices leave RF proximity with a client device as well as when a service is made unavailable on a device that is in RF proximity with the client device.
6. SDP shall provide for services, classes of services, and attributes of services to be uniquely identified.
7. SDP shall allow a client on one device to discover a service on another device without consulting a third device.
8. SDP should be suitable for use on devices of limited complexity.
9. SDP shall provide a mechanism to incrementally discover information about the services provided by a device. This is intended to minimize the quantity

of data that must be exchanged in order to determine that a particular service is not needed by a client.

10. SDP should support the caching of service discovery information by intermediary agents to improve the speed or efficiency of the discovery process.
11. SDP should be transport independent.
12. SDP shall function while using L2CAP as its transport protocol.
13. SDP shall permit the discovery and use of services that provide access to other service discovery protocols.
14. SDP shall support the creation and definition of new services without requiring registration with a central authority.

## 1.4 NON-REQUIREMENTS AND DEFERRED REQUIREMENTS

The Bluetooth SIG recognizes that the following capabilities are related to service discovery. These items are not addressed in SDP version 1.0. However, some may be addressed in future revisions of the specification.

1. SDP 1.0 does not provide access to services. It only provides access to information about services.
2. SDP 1.0 does not provide brokering of services.
3. SDP 1.0 does not provide for negotiation of service parameters.
4. SDP 1.0 does not provide for billing of service use.
5. SDP 1.0 does not provide the means for a client to control or change the operation of a service.
6. SDP 1.0 does not provide an event notification when services, or information about services, become unavailable.
7. SDP 1.0 does not provide an event notification when attributes of services are modified.
8. This specification does not define an application programming interface for SDP.
9. SDP 1.0 does not provide support for service agent functions such as service aggregation or service registration.

## 1.5 CONVENTIONS

### 1.5.1 Bit And Byte Ordering Conventions

When multiple bit fields are contained in a single byte and represented in a drawing in this specification, the more significant (high-order) bits are shown toward the left and less significant (low-order) bits toward the right.

Multiple-byte fields are drawn with the more significant bytes toward the left and the less significant bytes toward the right. Multiple-byte fields are transferred in network byte order. See Section 4.1 on page 223.

# 2 OVERVIEW

## 2.1 SDP CLIENT-SERVER INTERACTION

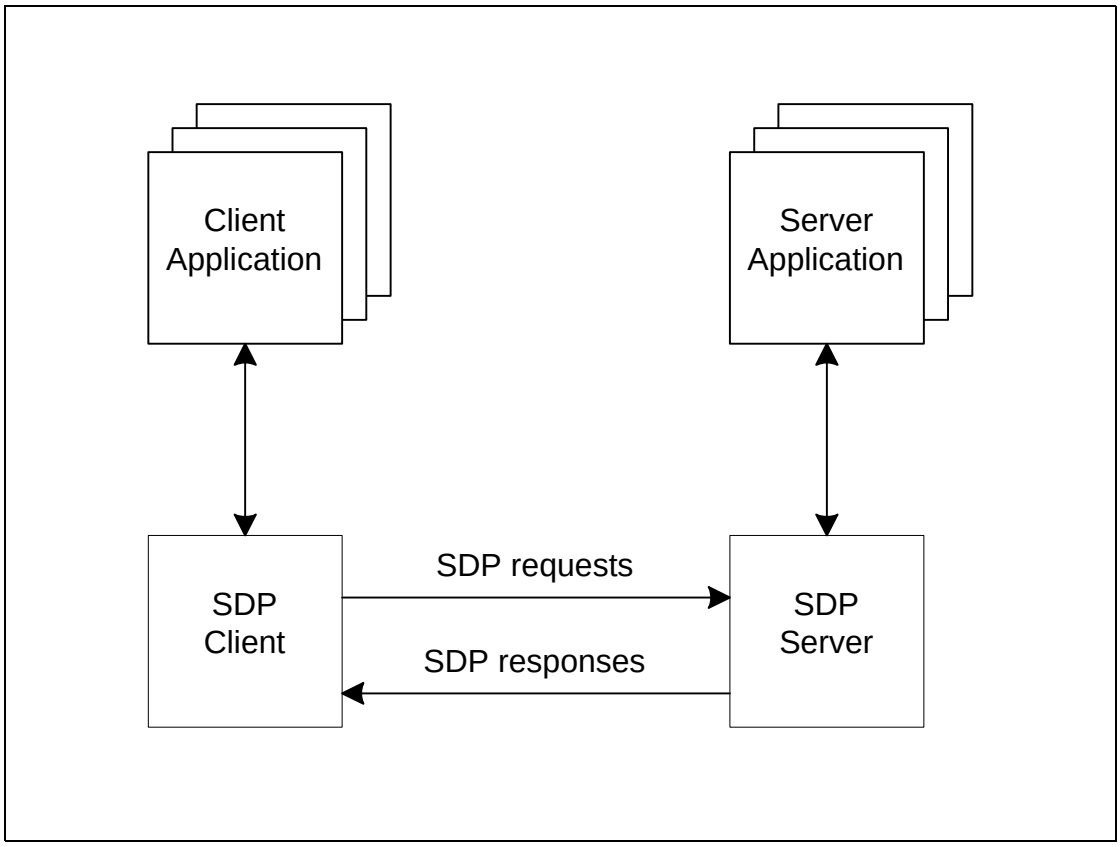



*Figure 2.1: SDP Client-Server Interaction*

The service discovery mechanism provides the means for client applications to discover the existence of services provided by server applications as well as the attributes of those services. The attributes of a service include the type or class of service offered and the mechanism or protocol information needed to utilize the service.

As far as the Service Discovery Protocol (SDP) is concerned, the configuration shown in Figure 2.1 can be simplified to that shown in Figure 2.2.

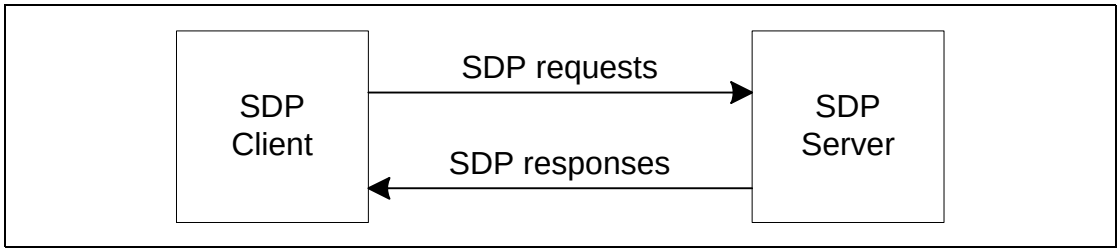



*Figure 2.2: Simplified SDP Client-Server Interaction*

SDP involves communication between an SDP server and an SDP client. The server maintains a list of service records that describe the characteristics of services associated with the server. Each service record contains information about a single service. A client can retrieve information from a service record maintained by the SDP server by issuing an SDP request.

If the client, or an application associated with the client, decides to use a service, it opens a separate connection to the service provider in order to utilize the service. SDP provides a mechanism for discovering services and their attributes (including associated service access protocols), but it does not provide a mechanism for utilizing those services (such as delivering the service access protocols).

There shall be a maximum of one active SDP server per Bluetooth device. If a Bluetooth device acts only as a client, it needs no SDP server. A single Bluetooth device can function both as an SDP server and as an SDP client. If multiple applications on a device provide services, an SDP server can act on behalf of those service providers to handle requests for information about the services that they provide.

Similarly, multiple client applications can utilize an SDP client to query servers on behalf of the client applications.

The set of SDP servers that are available to an SDP client will change dynamically based on the RF proximity of the servers to the client. When a server becomes available, a potential client must be notified by a means other than SDP so that the client can use SDP to query the server about its services. Similarly, when a server leaves proximity or becomes unavailable for any reason, there is no explicit notification via the service discovery protocol. However, the client can use SDP to poll the server and may infer that the server is not available if it no longer responds to requests.

Additional information regarding application interaction with SDP shall be contained in the Bluetooth Service Discovery Profile document.

## 2.2 SERVICE RECORD

A service is any entity that can provide information, perform an action, or control a resource on behalf of another entity. A service may be implemented as software, hardware, or a combination of hardware and software.

All of the information about a service that is maintained by an SDP server is contained within a single service record. The service record shall only be a list of service attributes.

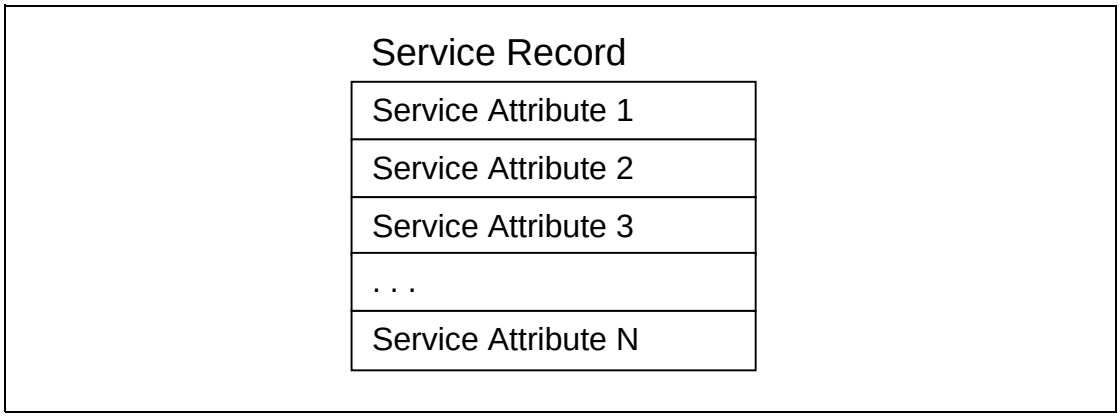



*Figure 2.3: Service Record*

A service record handle is a 32-bit number that shall uniquely identify each service record within an SDP server. It is important to note that, in general, each handle is unique only within each SDP server. If SDP server S1 and SDP server S2 both contain identical service records (representing the same service), the service record handles used to reference these identical service records are completely independent. The handle used to reference the service on S1 will be meaningless if presented to S2.

The service discovery protocol does not provide a mechanism for notifying clients when service records are added to or removed from an SDP server. While an L2CAP (Logical Link Control and Adaptation Protocol) connection is established to a server, a service record handle acquired from the server shall remain valid unless the service record it represents is removed. If a service is removed from the server, further requests to the server (during the L2CAP connection in which the service record handle was acquired) using the service's (now stale) record handle shall result in an error response indicating an invalid service record handle. An SDP server shall ensure that no service record handle values are re-used while an L2CAP connection remains established. Service record handles shall remain valid across successive L2CAP connections while the ServiceDatabaseState attribute value remains unchanged. Further, service record handles should remain valid until such time that the corresponding service is permanently removed or changes in an incompatible way. See the ServiceRecordState and ServiceDatabaseState attributes in Section 5 on page 237.

A device may have a service record with a service record handle of 0x00000000 representing the SDP server itself. This service record contains attributes for the SDP server and the protocol it supports. For example, one of its attributes is the list of SDP protocol versions supported by the server.

## 2.3 SERVICE ATTRIBUTE

Each service attribute describes a single characteristic of a service. Some examples of service attributes are:

| | |
|---|---|
| ServiceClassIDList | Identifies the type of service represented by a service record. In other words, the list of classes of which the service is an instance |
| ServiceID | Uniquely identifies a specific instance of a service |
| ProtocolDescriptorList | Specifies the protocol stack(s) that may be used to utilize a service |
| ProviderName | The textual name of the individual or organization that provides a service |
| IconURL | Specifies a URL that refers to an icon image that may be used to represent a service |
| ServiceName | A text string containing a human-readable name for the service |
| ServiceDescription | A text string describing the service |

See Section 5.1 on page 237, for attribute definitions that are common to all service records. Service providers can also define their own service attributes.

A service attribute consists of two components: an attribute ID and an attribute value.

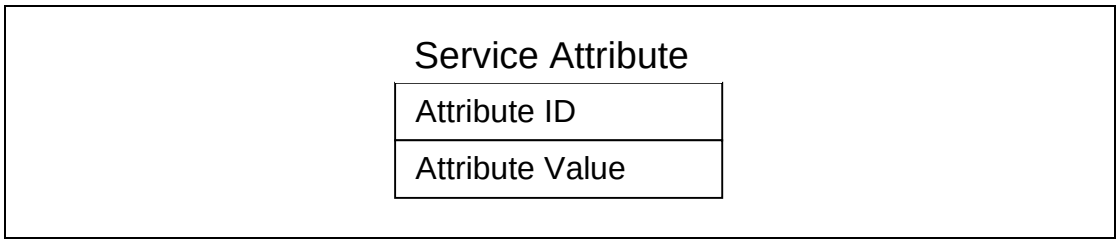



*Figure 2.4: Service Attribute*

### 2.3.1 Attribute ID

An attribute ID is a 16-bit unsigned integer that distinguishes each service attribute from other service attributes within a service record. The attribute ID also identifies the semantics of the associated attribute value.

A service class definition specifies each of the attribute IDs for a service class and assigns a meaning to the attribute value associated with each attribute ID. Each attribute ID is defined to be unique only within each service class.

All services belonging to a given service class assign the same meaning to each particular attribute ID. See Section 2.4 on page 215.

In the Service Discovery Protocol, an attribute ID is represented as a data element. See Section 3 on page 220.

### 2.3.2 Attribute Value

The attribute value is a variable length field whose meaning is determined by the attribute ID associated with it and by the service class of the service record in which the attribute is contained. In the Service Discovery Protocol, an attribute value is represented as a data element. (See Section 3 on page 220.) Generally, any type of data element is permitted as an attribute value, subject to the constraints specified in the service class definition that assigns an attribute ID to the attribute and assigns a meaning to the attribute value. See Section 5 on page 237, for attribute value examples.

## 2.4 SERVICE CLASS

Each service is an instance of a service class. The service class definition provides the definitions of all attributes contained in service records that represent instances of that class. Each attribute definition specifies the numeric value of the attribute ID, the intended use of the attribute value, and the format of the attribute value. A service record contains attributes that are specific to a service class as well as universal attributes that are common to all services.

Each service class is also assigned a unique identifier. This service class identifier is contained in the attribute value for the ServiceClassIDList attribute, and is represented as a UUID (see Section 2.5.1 on page 216). Since the format and meanings of many attributes in a service record are dependent on the service class of the service record, the ServiceClassIDList attribute is very important. Its value shall be examined or verified before any class-specific attributes are used. Since all of the attributes in a service record must conform to all of the service's classes, the service class identifiers contained in the ServiceClassIDList attribute are related. Typically, each service class is a subclass of another class whose identifier is contained in the list. A service subclass definition differs from its superclass in that the subclass contains additional attribute definitions that are specific to the subclass. The service class identifiers in the ServiceClassIDList attribute should be listed in order from the most specific class to the most general class.

When a new service class is defined that is a subclass of an existing service class, the new service class retains all of the attributes defined in its superclass. Additional attributes may be defined that are specific to the new service class. In other words, the mechanism for adding new attributes to some of the instances of an existing service class is to create a new service class that is a subclass of the existing service class.

### 2.4.1 A Printer Service Class Example

A color postscript printer with duplex capability might conform to 4 ServiceClass definitions and have a ServiceClassIDList with UUIDs (See Section 2.5.1 on page 216.) representing the following ServiceClasses:

DuplexColorPostscriptPrinterServiceClassID,
ColorPostscriptPrinterServiceClassID,
PostscriptPrinterServiceClassID,
PrinterServiceClassID

*Note that this example is only illustrative. This may not be a practical printer class hierarchy.*

## 2.5 SEARCHING FOR SERVICES

The Service Search transaction allows a client to retrieve the service record handles for particular service records based on the values of attributes contained within those service records. Once an SDP client has a service record handle, it can request the values of specific attributes.

The capability search for service records based on the values of arbitrary attributes is not provided. Rather, the capability is provided to search only for attributes whose values are Universally Unique Identifiers[1] (UUIDs). Important attributes of services that can be used to search for a service are represented as UUIDs.

### 2.5.1 UUID

A UUID is a universally unique identifier that is guaranteed to be unique across all space and all time. UUIDs can be independently created in a distributed fashion. No central registry of assigned UUIDs is required. A UUID is a 128-bit value.

To reduce the burden of storing and transferring 128-bit UUID values, a range of UUID values has been pre-allocated for assignment to often-used, registered purposes. The first UUID in this pre-allocated range is known as the Bluetooth Base UUID and has the value 00000000-0000-1000-8000-00805F9B34FB, from the Bluetooth Assigned Numbers document. UUID values in the pre-allocated range have aliases that are represented as 16-bit or 32-bit values. These aliases are often called 16-bit and 32-bit UUIDs, but it is important to note that each actually represents a 128-bit UUID value.

The full 128-bit value of a 16-bit or 32-bit UUID may be computed by a simple arithmetic operation.

$128\_bit\_value = 16\_bit\_value * 2^{96} + Bluetooth\_Base\_UUID$

$128\_bit\_value = 32\_bit\_value * 2^{96} + Bluetooth\_Base\_UUID$

1. The format of UUIDs is defined by the International Organization for Standardization in ISO/IEC 11578:1996. “Information technology – Open Systems Interconnection – Remote Procedure Call (RPC)”

A 16-bit UUID may be converted to 32-bit UUID format by zero-extending the 16-bit value to 32-bits. An equivalent method is to add the 16-bit UUID value to a zero-valued 32-bit UUID.

Note that two 16-bit UUIDs may be compared directly, as may two 32-bit UUIDs or two 128-bit UUIDs. If two UUIDs of differing sizes are to be compared, the shorter UUID must be converted to the longer UUID format before comparison.

### 2.5.2 Service Search Patterns

A service search pattern is a list of UUIDs used to locate matching service records. A service search pattern matches a service record if each and every UUID in the service search pattern is contained within any of the service record's attribute values. The UUIDs need not be contained within any specific attributes or in any particular order within the service record. The service search pattern matches if the UUIDs it contains constitute a subset of the UUIDs in the service record's attribute values. The only time a service search pattern does not match a service record is if the service search pattern contains at least one UUID that is not contained within the service record's attribute values. Note also that a valid service search pattern must contain at least one UUID.

## 2.6 BROWSING FOR SERVICES

Normally, a client searches for services based on some desired characteristic(s) (represented by a UUID) of the services. However, there are times when it is desirable to discover which types of services are described by an SDP server's service records without any a priori information about the services. This process of looking for any offered services is termed browsing. In SDP, the mechanism for browsing for services is based on an attribute shared by all service classes. This attribute is called the BrowseGroupList attribute. The value of this attribute contains a list of UUIDs. Each UUID represents a browse group with which a service may be associated for the purpose of browsing.

When a client desires to browse an SDP server's services, it creates a service search pattern containing the UUID that represents the root browse group. All services that may be browsed at the top level are made members of the root browse group by having the root browse group's UUID as a value within the BrowseGroupList attribute.

Normally, if an SDP server has relatively few services, all of its services will be placed in the root browse group. However, the services offered by an SDP server may be organized in a browse group hierarchy, by defining additional browse groups below the root browse group. Each of these additional browse groups is described by a service record with a service class of BrowseGroupDescriptor.

A browse group descriptor service record defines a new browse group by means of its Group ID attribute. In order for a service contained in one of these newly defined browse groups to be browseable, the browse group descriptor service record that defines the new browse group must in turn be browseable. The hierarchy of browseable services that is provided by the use of browse group descriptor service records allows the services contained in an SDP server to be incrementally browsed and is particularly useful when the SDP server contains many service records.

### 2.6.1 Example Service Browsing Hierarchy

Here is a fictitious service browsing hierarchy that illuminates the manner in which browse group descriptors are used. Browse group descriptor service records are identified with (G); other service records with (S).

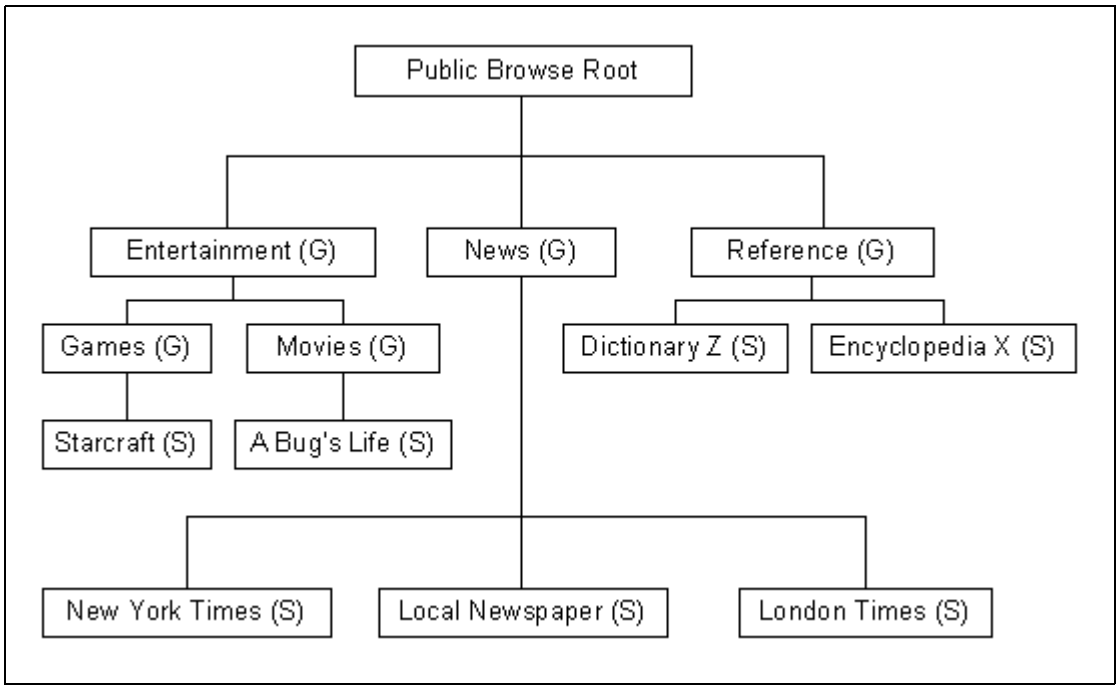



*Figure 2.5: Service Browsing Hierarchy*

This table shows the services records and service attributes necessary to implement the browse hierarchy.

| Service Name | Service Class | Attribute Name | Attribute Value |
|---|---|---|---|
| Entertainment | BrowseGroupDescriptor | BrowseGroupList | PublicBrowseRoot |
| | | GroupID | EntertainmentID |
| News | BrowseGroupDescriptor | BrowseGroupList | PublicBrowseRoot |
| | | GroupID | NewsID |
| Reference | BrowseGroupDescriptor | BrowseGroupList | PublicBrowseRoot |
| | | GroupID | ReferenceID |
| Games | BrowseGroupDescriptor | BrowseGroupList | EntertainmentID |
| | | GroupID | GamesID |
| Movies | BrowseGroupDescriptor | BrowseGroupList | EntertainmentID |
| | | GroupID | MoviesID |
| Starcraft | Video Game Class ID | BrowseGroupList | GamesID |
| A Bug's Life | Movie Class ID | BrowseGroupList | MovieID |
| Dictionary Z | Dictionary Class ID | BrowseGroupList | ReferenceID |
| Encyclopedia X | Encyclopedia Class ID | BrowseGroupList | ReferenceID |
| New York Times | Newspaper ID | BrowseGroupList | NewspaperID |
| London Times | Newspaper ID | BrowseGroupList | NewspaperID |
| Local Newspaper | Newspaper ID | BrowseGroupList | NewspaperID |

*Table 2.1: Service Browsing Hierarchy*

# 3 DATA REPRESENTATION

Attribute values can contain information of various types with arbitrary complexity; thus enabling an attribute list to be generally useful across a wide variety of service classes and environments.

SDP defines a simple mechanism to describe the data contained within an attribute ID, attribute ID range, and attribute value. The primitive construct used is the data element.

## 3.1 DATA ELEMENT

A data element is a typed data representation. It consists of two fields: a header field and a data field. The header field, in turn, is composed of two parts: a type descriptor and a size descriptor. The data is a sequence of bytes whose length is specified in the size descriptor (described in Section 3.3 on page 221) and whose meaning is (partially) specified by the type descriptor.

## 3.2 DATA ELEMENT TYPE DESCRIPTOR

A data element type is represented as a 5-bit type descriptor. The type descriptor is contained in the most significant (high-order) 5 bits of the first byte of the data element header. The following types have been defined.

| Type Descriptor Value | Valid Size Descriptor Values | Type Description |
|---|---|---|
| 0 | 0 | Nil, the null type |
| 1 | 0, 1, 2, 3, 4 | Unsigned Integer |
| 2 | 0, 1, 2, 3, 4 | Signed twos-complement integer |
| 3 | 1, 2, 4 | UUID, a universally unique identifier |
| 4 | 5, 6, 7 | Text string |
| 5 | 0 | Boolean[1] |
| 6 | 5, 6, 7 | Data element sequence, a data element whose data field is a sequence of data elements |
| 7 | 5, 6, 7 | Data element alternative, data element whose data field is a sequence of data elements from which one data element is to be selected. |
| 8 | 5, 6, 7 | URL, a uniform resource locator |
| 9-31 | | Reserved |

*Table 3.1: Data Element*

1. False is represented by the value 0, and true is represented by the value 1. However, to maximize interoperability, any non-zero value received must be accepted as representing true.

## 3.3 DATA ELEMENT SIZE DESCRIPTOR

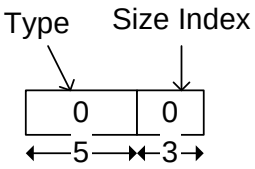
The data element size descriptor is represented as a 3-bit size index followed by 0, 8, 16, or 32 bits. The size index is contained in the least significant (low-order) 3 bits of the first byte of the data element header. The size index is encoded as follows.

| Size Index | Additional bits | Data Size |
|---|---|---|
| 0 | 0 | 1 byte. Exception: if the data element type is nil, the data size is 0 bytes. |
| 1 | 0 | 2 bytes |
| 2 | 0 | 4 bytes |
| 3 | 0 | 8 bytes |
| 4 | 0 | 16 bytes |
| 5 | 8 | The data size is contained in the additional 8 bits, which are interpreted as an unsigned integer. |
| 6 | 16 | The data size is contained in the additional 16 bits, which are interpreted as an unsigned integer. |
| 7 | 32 | The data size is contained in the additional 32 bits, which are interpreted as an unsigned integer. |

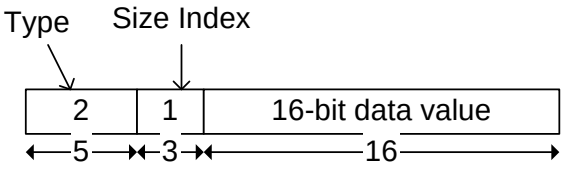
*Table 3.2: Data Element Size*

## 3.4 DATA ELEMENT EXAMPLES

Nil is represented as:




A 16-bit signed integer is represented as:




The 3 character ASCII string "Hat" is represented as:

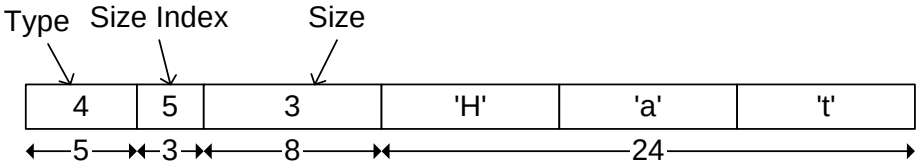



*Figure 3.1: Data Element*

# 4 PROTOCOL DESCRIPTION

SDP is a simple protocol with minimal requirements on the underlying transport. It can function over a reliable packet transport (or even unreliable, if the client implements timeouts and repeats requests as necessary).

SDP uses a request/response model where each transaction consists of one request protocol data unit (PDU) and one response PDU. In the case where SDP is used with the Bluetooth L2CAP transport protocol, no more than one SDP request PDU per connection to a given SDP server may be outstanding at a given instant. In other words, a client does not issue a further request to a server prior to receiving a response to the current request before issuing another request on the same L2CAP connection. Limiting SDP to sending one unacknowledged request PDU provides a simple form of flow control.

The protocol examples found in Appendix B – Example SDP Transactions, could be helpful in understanding the protocol transactions.

## 4.1 TRANSFER BYTE ORDER

The service discovery protocol shall transfer multiple-byte fields in standard network byte order (Big Endian), with more significant (high-order) bytes being transferred before less-significant (low-order) bytes.

## 4.2 PROTOCOL DATA UNIT FORMAT

Every SDP PDU consists of a PDU header followed by PDU-specific parameters. The header contains three fields: a PDU ID, a Transaction ID, and a ParameterLength. Each of these header fields is described here. Parameters may include a continuation state parameter, described below; PDU-specific parameters for each PDU type are described later in separate PDU descriptions.

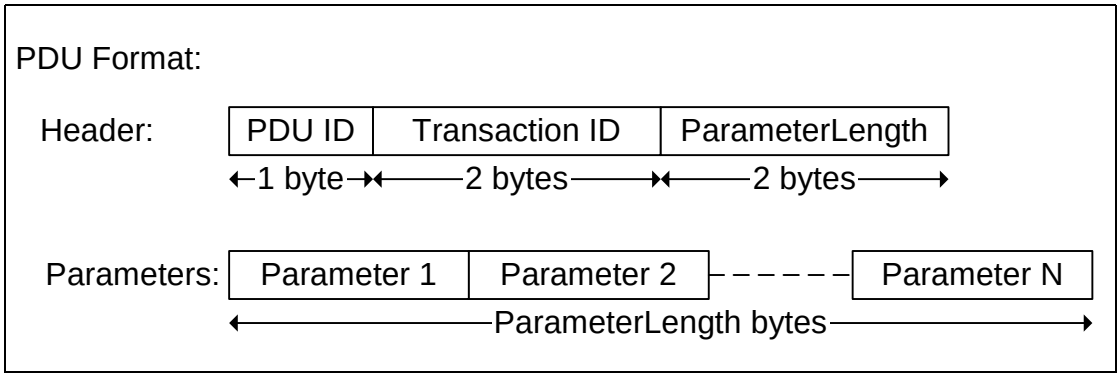



*Figure 4.1: Protocol Data Unit Format*

*PDU ID:* *Size: 1 Byte*

| Value | Parameter Description |
|---|---|
| N | The PDU ID field identifies the type of PDU. I.e. its meaning and the specific parameters. |
| 0x00 | Reserved |
| 0x01 | SDP_ErrorResponse |
| 0x02 | SDP_ServiceSearchRequest |
| 0x03 | SDP_ServiceSearchResponse |
| 0x04 | SDP_ServiceAttributeRequest |
| 0x05 | SDP_ServiceAttributeResponse |
| 0x06 | SDP_ServiceSearchAttributeRequest |
| 0x07 | SDP_ServiceSearchAttributeResponse |
| 0x07-0xFF | Reserved |

*TransactionID:* *Size: 2 Bytes*

| Value | Parameter Description |
|---|---|
| N | The TransactionID field uniquely identifies request PDUs and is used to match response PDUs to request PDUs. The SDP client can choose any value for a request's TransactionID provided that it is different from all outstanding requests. The TransactionID value in response PDUs is required to be the same as the request that is being responded to.<br>Range: 0x0000 – 0xFFFF |

*ParameterLength:* *Size: 2 Bytes*

| Value | Parameter Description |
|---|---|
| N | The ParameterLength field specifies the length (in bytes) of all parameters contained in the PDU.<br>Range: 0x0000 – 0xFFFF |

## 4.3 PARTIAL RESPONSES AND CONTINUATION STATE

Some SDP requests may require responses that are larger than can fit in a single response PDU. In this case, the SDP server shall generate a partial response along with a continuation state parameter. The continuation state parameter can be supplied by the client in a subsequent request to retrieve the next portion of the complete response. The continuation state parameter is a variable length field whose first byte contains the number of additional bytes of continuation information in the field. The format of the continuation information is not standardized among SDP servers. Each continuation state parameter is meaningful only to the SDP server that generated it.

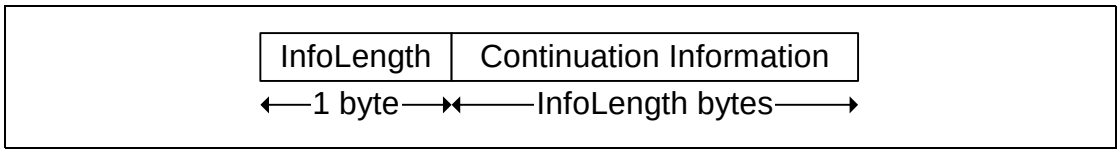



*Figure 4.2: Continuation State Format*

After a client receives a partial response and the accompanying continuation state parameter, it can re-issue the original request (with a new transaction ID) and include the continuation state in the new request indicating to the server that the remainder of the original response is desired. The maximum allowable value of the InfoLength field is 16 (0x10).

Note that, unless otherwise stated in a PDU response specification, an SDP server can split a response at any arbitrary boundary when it generates a partial response. The SDP server may select the boundary based on the contents of the reply, but is not required to do so. Therefore, length fields give the lengths as measured in the complete assembled record, not the length of the fields in the partial segment. When a service record is segmented into partial responses all attribute list values are relative to the complete record, not relevant to the partial record.

## 4.4 ERROR HANDLING

Each transaction consists of a request and a response PDU. Generally, each type of request PDU has a corresponding type of response PDU. However, if the server determines that a request is improperly formatted or for any reason the server cannot respond with the appropriate PDU type, it shall respond with an SDP_ErrorResponse PDU.

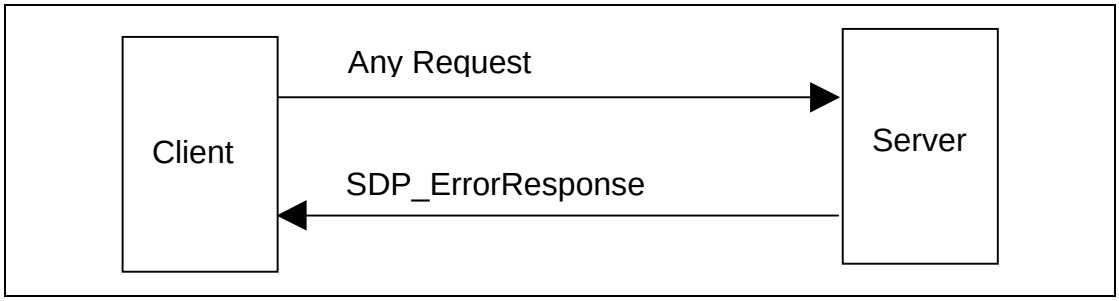



*Figure 4.3: Error Handling*

### 4.4.1 SDP_ErrorResponse PDU

| PDU Type | PDU ID | Parameters |
|---|---|---|
| SDP_ErrorResponse | 0x01 | ErrorCode |

**Description:**

The SDP server generates this PDU type in response to an improperly formatted request PDU or when the SDP server, for whatever reason, cannot generate an appropriate response PDU.

**PDU Parameters:**

*ErrorCode:* *Size: 2 Bytes*

| Value | Parameter Description |
|---|---|
| N | The ErrorCode identifies the reason that an SDP_ErrorResponse PDU was generated. |
| 0x0000 | Reserved |
| 0x0001 | Invalid/unsupported SDP version |
| 0x0002 | Invalid Service Record Handle |
| 0x0003 | Invalid request syntax |
| 0x0004 | Invalid PDU Size |
| 0x0005 | Invalid Continuation State |
| 0x0006 | Insufficient Resources to satisfy Request |
| 0x0007-0xFFFF | Reserved |

Note: Invalid PDU size should be used, for example, if an incoming request PDU's length is inconsistent with the specification of that request PDU or the length parameter of an incoming request PDU is inconsistent with that request PDU's actual contents.

## 4.5 SERVICESEARCH TRANSACTION




*Figure 4.4: ServiceSearch Transaction*

### 4.5.1 SDP_ServiceSearchRequest PDU

| PDU Type | PDU ID | Parameters |
|---|---|---|
| SDP_ServiceSearchRequest | 0x02 | ServiceSearchPattern, MaximumServiceRecordCount, ContinuationState |

**Description:**

The SDP client generates an SDP_ServiceSearchRequest to locate service records that match the service search pattern given as the first parameter of the PDU. Upon receipt of this request, the SDP server shall examine its service record data base and return an SDP_ServiceSearchResponse containing the service record handles of service records within its service record database that match the given service search pattern, or an appropriate error response.

Note that no mechanism is provided to request information for all service records. However, see Section 2.6 on page 217 for a description of a mechanism that permits browsing for non-specific services without a priori knowledge of the services.

**PDU Parameters:**

*ServiceSearchPattern:* *Size: Varies*

| Value | Parameter Description |
|---|---|
| Data Element Sequence | The ServiceSearchPattern is a data element sequence where each element in the sequence is a UUID. The sequence shall contain at least one UUID. The maximum number of UUIDs in the sequence is 12[1]. The list of UUIDs constitutes a service search pattern. |

1. The value of 12 has been selected as a compromise between the scope of a service search and the size of a search request PDU. It is not expected that more than 12 UUIDs will be useful in a service search pattern.

*MaximumServiceRecordCount:* *Size: 2 Bytes*

| Value | Parameter Description |
|---|---|
| N | MaximumServiceRecordCount is a 16-bit count specifying the maximum number of service record handles to be returned in the response(s) to this request. The SDP server shall not return more handles than this value specifies. If more than N service records match the request, the SDP server determines which matching service record handles to return in the response(s).<br>Range: 0x0001-0xFFFF |

*ContinuationState:* *Size: 1 to 17 Bytes*

| Value | Parameter Description |
|---|---|
| Continuation State | ContinuationState consists of an 8-bit count, N, of the number of bytes of continuation state information, followed by the N bytes of continuation state information that were returned in a previous response from the server. N is required to be less than or equal to 16. If no continuation state is to be provided in the request, N is set to 0. |

### 4.5.2 SDP_ServiceSearchResponse PDU

| PDU Type | PDU ID | Parameters |
|---|---|---|
| SDP_ServiceSearchResponse | 0x03 | TotalServiceRecordCount, CurrentServiceRecordCount, ServiceRecordHandleList, ContinuationState |

**Description:**

The SDP server generates an SDP_ServiceSearchResponse upon receipt of a valid SDP_ServiceSearchRequest. The response contains a list of service record handles for service records that match the service search pattern given in the request. If a partial response is generated, it shall contain an integral number of complete service record handles; a service record handle value shall not be split across multiple PDUs.

**PDU Parameters:**

*TotalServiceRecordCount:* *Size: 2 Bytes*

| Value | Parameter Description |
|---|---|
| N | The TotalServiceRecordCount is an integer containing the number of service records that match the requested service search pattern. If no service records match the requested service search pattern, this parameter is set to 0. N should never be larger than the MaximumServiceRecordCount value specified in the SDP_ServiceSearchRequest. When multiple partial responses are used, each partial response contains the same value for TotalServiceRecordCount.<br>Range: 0x0000-0xFFFF |

*CurrentServiceRecordCount:* *Size: 2 Bytes*

| Value | Parameter Description |
|---|---|
| N | The CurrentServiceRecordCount is an integer indicating the number of service record handles that are contained in the next parameter. If no service records match the requested service search pattern, this parameter is set to 0. N should never be larger than the TotalServiceRecordCount value specified in the current response.<br>Range: 0x0000-0xFFFF |

*ServiceRecordHandleList:* *Size: (CurrentServiceRecordCount*4) Bytes*

| Value | Parameter Description |
|---|---|
| List of 32-bit handles | The ServiceRecordHandleList contains a list of service record handles. The number of handles in the list is given in the CurrentServiceRecordCount parameter. Each of the handles in the list refers to a service record that matches the requested service search pattern. Note that this list of service record handles does not have the format of a data element. It contains no header fields, only the 32-bit service record handles. |

*ContinuationState:* *Size: 1 to 17 Bytes*

| Value | Parameter Description |
|---|---|
| Continuation State | ContinuationState consists of an 8-bit count, N, of the number of bytes of continuation state information, followed by the N bytes of continuation information. If the current response is complete, this parameter consists of a single byte with the value 0. If a partial response is contained in the PDU, the ContinuationState parameter may be supplied in a subsequent request to retrieve the remainder of the response. |

## 4.6 SERVICEATTRIBUTE TRANSACTION

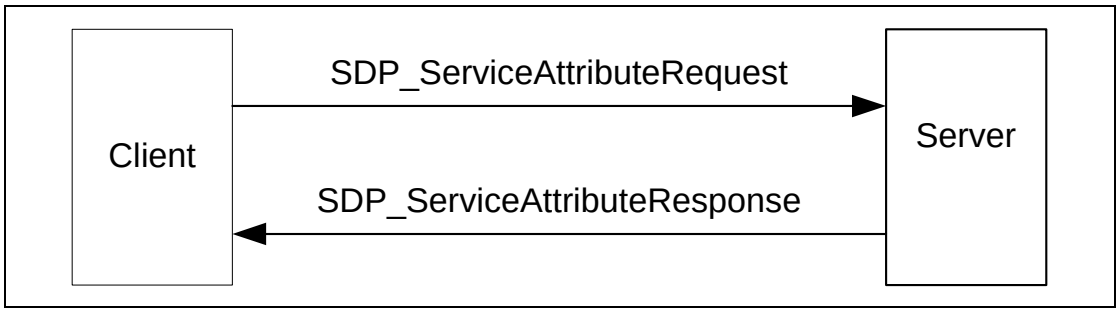



*Figure 4.5: ServiceAttribute Transaction*

### 4.6.1 SDP_ServiceAttributeRequest PDU

| PDU Type | PDU ID | Parameters |
|---|---|---|
| SDP_ServiceAttributeRequest | 0x04 | ServiceRecordHandle, MaximumAttributeByteCount, AttributeIDList, ContinuationState |

**Description:**

The SDP client generates an SDP_ServiceAttributeRequest to retrieve specified attribute values from a specific service record. The service record handle of the desired service record and a list of desired attribute IDs to be retrieved from that service record are supplied as parameters.

**Command Parameters:**

*ServiceRecordHandle:* *Size: 4 Bytes*

| Value | Parameter Description |
|---|---|
| 32-bit handle | The ServiceRecordHandle parameter specifies the service record from which attribute values are to be retrieved. The handle is obtained via a previous SDP_ServiceSearch transaction. |

*MaximumAttributeByteCount:* *Size: 2 Bytes*

| Value | Parameter Description |
|---|---|
| N | MaximumAttributeByteCount specifies the maximum number of bytes of attribute data to be returned in the response to this request. The SDP server shall not return more than N bytes of attribute data in the response PDU. If the requested attributes require more than N bytes, the SDP server determines how to segment the list. In this case the client may request each successive segment by issuing a request containing the continuation state that was returned in the previous response PDU. Note that in the case where multiple response PDUs are needed to return the attribute data, MaximumAttributeByteCount specifies the maximum size of the portion of the attribute data contained in each response PDU.<br>Range: 0x0007-0xFFFF |

*AttributeIDList:* *Size: Varies*

| Value | Parameter Description |
|---|---|
| Data Element Sequence | The AttributeIDList is a data element sequence where each element in the list is either an attribute ID or a range of attribute IDs. Each attribute ID is encoded as a 16-bit unsigned integer data element. Each attribute ID range is encoded as a 32-bit unsigned integer data element, where the high order 16 bits are interpreted as the beginning attribute ID of the range and the low order 16 bits are interpreted as the ending attribute ID of the range. The attribute IDs contained in the AttributeIDList shall be listed in ascending order without duplication of any attribute ID values. Note that all attributes can be requested by specifying a range of 0x0000-0xFFFF. |

*ContinuationState:* *Size: 1 to 17 Bytes*

| Value | Parameter Description |
|---|---|
| Continuation State | ContinuationState consists of an 8-bit count, N, of the number of bytes of continuation state information, followed by the N bytes of continuation state information that were returned in a previous response from the server. N shall be less than or equal to 16. If no continuation state is to be provided in the request, N shall be set to 0. |

### 4.6.2 SDP_ServiceAttributeResponse PDU

| PDU Type | PDU ID | Parameters |
|---|---|---|
| SDP_ServiceAttributeResponse | 0x05 | AttributeListByteCount, AttributeList, ContinuationState |

**Description:**

The SDP server generates an SDP_ServiceAttributeResponse upon receipt of a valid SDP_ServiceAttributeRequest. The response contains a list of attributes (both attribute ID and attribute value) from the requested service record.

**PDU Parameters:**

*AttributeListByteCount:* *Size: 2 Bytes*

| Value | Parameter Description |
|---|---|
| N | The AttributeListByteCount contains a count of the number of bytes in the AttributeList parameter. N shall never be larger than the MaximumAttributeByteCount value specified in the SDP_ServiceAttributeRequest.<br>Range: 0x0002-0xFFFF |

*AttributeList:* *Size: AttributeListByteCount*

| Value | Parameter Description |
|---|---|
| Data Element Sequence | The AttributeList is a data element sequence containing attribute IDs and attribute values. The first element in the sequence contains the attribute ID of the first attribute to be returned. The second element in the sequence contains the corresponding attribute value. Successive pairs of elements in the list contain additional attribute ID and value pairs. Only attributes that have non-null values within the service record and whose attribute IDs were specified in the SDP_ServiceAttributeRequest are contained in the AttributeList. Neither an attribute ID nor an attribute value is placed in the AttributeList for attributes in the service record that have no value. The attributes are listed in ascending order of attribute ID value. |

*ContinuationState:* *Size: 1 to 17 Bytes*

| Value | Parameter Description |
|---|---|
| Continuation State | ContinuationState consists of an 8-bit count, N, of the number of bytes of continuation state information, followed by the N bytes of continuation information. If the current response is complete, this parameter consists of a single byte with the value 0. If a partial response is given, the ContinuationState parameter may be supplied in a subsequent request to retrieve the remainder of the response. |

If a partial response is generated, the response may be arbitrarily segmented into multiple PDUs (subject to the constraint imposed by the allowed range of values for the AttributeListByteCount parameter). Attributes in partial responses are not required to be integral.

## 4.7 SERVICESEARCHATTRIBUTE TRANSACTION

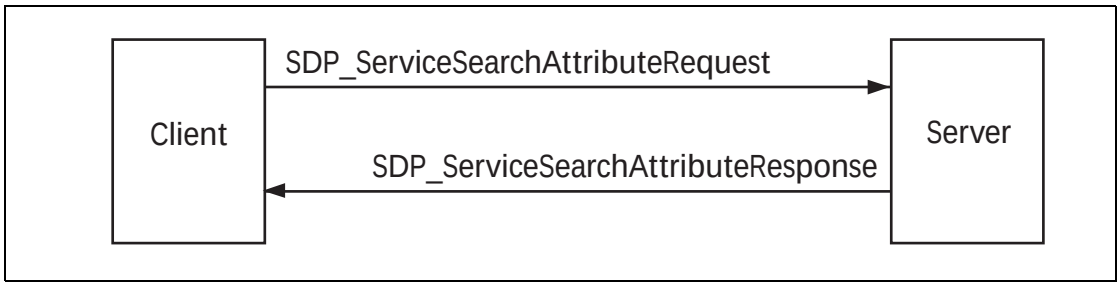



*Figure 4.6: ServiceSearchAttribute Transaction*

### 4.7.1 SDP_ServiceSearchAttributeRequest PDU

| PDU Type | PDU ID | Parameters |
|---|---|---|
| SDP_ServiceSearchAttributeRequest | 0x06 | ServiceSearchPattern, MaximumAttributeByteCount, AttributeIDList, ContinuationState |

**Description:**

The SDP_ServiceSearchAttributeRequest transaction combines the capabilities of the SDP_ServiceSearchRequest and the SDP_ServiceAttributeRequest into a single request. As parameters, it contains both a service search pattern and a list of attributes to be retrieved from service records that match the service search pattern. The SDP_ServiceSearchAttributeRequest and its response are more complex and can require more bytes than separate SDP_ServiceSearch and SDP_ServiceAttribute transactions. However, using SDP_ServiceSearchAttributeRequest can reduce the total number of SDP transactions, particularly when retrieving multiple service records.

Note that the service record handle for each service record is contained in the ServiceRecordHandle attribute of that service and may be requested along with other attributes.

### PDU Parameters:

*ServiceSearchPattern:* *Size: Varies*

| Value | Parameter Description |
|---|---|
| Data Element Sequence | The ServiceSearchPattern is a data element sequence where each element in the sequence is a UUID. The sequence must contain at least one UUID. The maximum number of UUIDs in the sequence is 12[1]. The list of UUIDs constitutes a service search pattern. |

1. The value of 12 has been selected as a compromise between the scope of a service search and the size of a search request PDU. It is not expected that more than 12 UUIDs will be useful in a service search pattern.

*MaximumAttributeByteCount:* *Size: 2 Bytes*

| Value | Parameter Description |
|---|---|
| N | MaximumAttributeByteCount specifies the maximum number of bytes of attribute data to be returned in the response to this request. The SDP server shall not return more than N bytes of attribute data in the response PDU. If the requested attributes require more than N bytes, the SDP server determines how to segment the list. In this case the client may request each successive segment by issuing a request containing the continuation state that was returned in the previous response PDU. Note that in the case where multiple response PDUs are needed to return the attribute data, MaximumAttributeByteCount specifies the maximum size of the portion of the attribute data contained in each response PDU.<br>Range: 0x0007-0xFFFF |

*AttributeIDList:* *Size: Varies*

| Value | Parameter Description |
|---|---|
| Data Element Sequence | The AttributeIDList is a data element sequence where each element in the list is either an attribute ID or a range of attribute IDs. Each attribute ID is encoded as a 16-bit unsigned integer data element. Each attribute ID range is encoded as a 32-bit unsigned integer data element, where the high order 16 bits are interpreted as the beginning attribute ID of the range and the low order 16 bits are interpreted as the ending attribute ID of the range. The attribute IDs contained in the AttributeIDList shall be listed in ascending order without duplication of any attribute ID values. Note that all attributes may be requested by specifying a range of 0x0000-0xFFFF. |

*ContinuationState:* *Size: 1 to 17 Bytes*

| Value | Parameter Description |
|---|---|
| Continuation State | ContinuationState consists of an 8-bit count, N, of the number of bytes of continuation state information, followed by the N bytes of continuation state information that were returned in a previous response from the server. N shall be less than or equal to 16. If no continuation state is to be provided in the request, N shall set to 0. |

### 4.7.2 SDP_ServiceSearchAttributeResponse PDU

| PDU Type | PDU ID | Parameters |
|---|---|---|
| SDP_ServiceSearchAttributeResponse | 0x07 | AttributeListsByteCount, AttributeLists, ContinuationState |

**Description:**

The SDP server generates an SDP_ServiceSearchAttributeResponse upon receipt of a valid SDP_ServiceSearchAttributeRequest. The response contains a list of attributes (both attribute ID and attribute value) from the service records that match the requested service search pattern.

**PDU Parameters:**

*AttributeListsByteCount:* *Size: 2 Bytes*

| Value | Parameter Description |
|---|---|
| N | The AttributeListsByteCount contains a count of the number of bytes in the AttributeLists parameter. N shall never be larger than the MaximumAttributeByteCount value specified in the SDP_ServiceSearchAttributeRequest.<br>Range: 0x0002-0xFFFF |

*AttributeLists:* *Size: Varies*

| Value | Parameter Description |
|---|---|
| Data Element Sequence | The AttributeLists is a data element sequence where each element in turn is a data element sequence representing an attribute list. Each attribute list contains attribute IDs and attribute values from one service record. The first element in each attribute list contains the attribute ID of the first attribute to be returned for that service record. The second element in each attribute list contains the corresponding attribute value. Successive pairs of elements in each attribute list contain additional attribute ID and value pairs. Only attributes that have non-null values within the service record and whose attribute IDs were specified in the SDP_ServiceSearchAttributeRequest are contained in the AttributeLists. Neither an attribute ID nor attribute value is placed in AttributeLists for attributes in the service record that have no value. Within each attribute list, the attributes are listed in ascending order of attribute ID value. |

*ContinuationState:* *Size: 1 to 17 Bytes*

| Value | Parameter Description |
|---|---|
| Continuation State | ContinuationState consists of an 8-bit count, N, of the number of bytes of continuation state information, followed by the N bytes of continuation information. If the current response is complete, this parameter consists of a single byte with the value 0. If a partial response is given, the ContinuationState parameter may be supplied in a subsequent request to retrieve the remainder of the response. |

If a partial response is generated, the response may be arbitrarily segmented into multiple PDUs (subject to the constraint imposed by the allowed range of values for the AttributeListByteCount parameter). Attributes in partial responses are not required to be integral.

# 5 SERVICE ATTRIBUTE DEFINITIONS

The service classes and attributes contained in this document are necessarily a partial list of the service classes and attributes supported by SDP. Only service classes that directly support the SDP server are included in this document. Additional service classes will be defined in other documents and possibly in future revisions of this document. Also, it is expected that additional attributes will be discovered that are applicable to a broad set of services; these may be added to the list of Universal attributes in future revisions of this document.

## 5.1 UNIVERSAL ATTRIBUTE DEFINITIONS

Universal attributes are those service attributes whose definitions are common to all service records. Note that this does not mean that every service record contains values for all of these service attributes. However, if a service record has a service attribute with an attribute ID allocated to a universal attribute, the attribute value shall conform to the universal attribute's definition.

Only two attributes are required to exist in every service record instance. They are the ServiceRecordHandle (attribute ID 0x0000) and the ServiceClassIDList (attribute ID 0x0001). All other service attributes are optional within a service record.

### 5.1.1 ServiceRecordHandle Attribute

| Attribute Name | Attribute ID | Attribute Value Type |
|---|---|---|
| ServiceRecordHandle | 0x0000 | 32-bit unsigned integer |

**Description:**

A service record handle is a 32-bit number that uniquely identifies each service record within an SDP server. It is important to note that, in general, each handle is unique only within each SDP server. If SDP server S1 and SDP server S2 both contain identical service records (representing the same service), the service record handles used to reference these identical service records are completely independent. The handle used to reference the service on S1 will, in general, be meaningless if presented to S2. Service record handle values 0x00000001-0x0000FFFF are reserved.

### 5.1.2 ServiceClassIDList Attribute

| Attribute Name | Attribute ID | Attribute Value Type |
|---|---|---|
| ServiceClassIDList | 0x0001 | Data Element Sequence |

**Description:**

The ServiceClassIDList attribute consists of a data element sequence in which each data element is a UUID representing the service classes that a given service record conforms to. The UUIDs shall be listed in order from the most specific class to the most general class unless overridden by a profile specification. The ServiceClassIDList shall contain at least one service class UUID.

### 5.1.3 ServiceRecordState Attribute

| Attribute Name | Attribute ID | Attribute Value Type |
|---|---|---|
| ServiceRecordState | 0x0002 | 32-bit unsigned integer |

**Description:**

The ServiceRecordState is a 32-bit integer that is used to facilitate caching of Service Attributes. If this attribute is contained in a service record, its value shall bechanged when any other attribute value is added to, deleted from or changed within the service record. This permits a client to check the value of this single attribute. If its value has not changed since it was last checked, the client knows that no other attribute values within the service record have changed.

### 5.1.4 ServiceID Attribute

| Attribute Name | Attribute ID | Attribute Value Type |
|---|---|---|
| ServiceID | 0x0003 | UUID |

**Description:**

The ServiceID is a UUID that universally and uniquely identifies the service instance described by the service record. This service attribute is particularly useful if the same service is described by service records in more than one SDP server.

### 5.1.5 ProtocolDescriptorList Attribute

| Attribute Name | Attribute ID | Attribute Value Type |
|---|---|---|
| ProtocolDescriptorList | 0x0004 | Data Element Sequence or Data Element Alternative |

**Description:**

The ProtocolDescriptorList attribute describes one or more protocol stacks that can be used to gain access to the service described by the service record.

If the ProtocolDescriptorList describes a single stack, it takes the form of a data element sequence in which each element of the sequence is a protocol descriptor. Each protocol descriptor is, in turn, a data element sequence whose first element is a UUID identifying the protocol and whose successive elements are protocol-specific parameters. Potential protocol-specific parameters are a protocol version number and a connection-port number. The protocol descriptors are listed in order from the lowest layer protocol to the highest layer protocol used to gain access to the service.

If it is possible for more than one kind of protocol stack to be used to gain access to the service, the ProtocolDescriptorList takes the form of a data element alternative where each member is a data element sequence as described in the previous paragraph.

**Protocol Descriptors**

A protocol descriptor identifies a communications protocol and provides protocol-specific parameters. A protocol descriptor is represented as a data element sequence. The first data element in the sequence shall be the UUID that identifies the protocol. Additional data elements optionally provide protocol-specific information, such as the L2CAP protocol/service multiplexer (PSM) and the RFCOMM server channel number (CN) shown below.

**ProtocolDescriptorList Examples**

These examples are intended to be illustrative. The parameter formats for each protocol are not defined within this specification.

In the first two examples, it is assumed that a single RFCOMM instance exists on top of the L2CAP layer. In this case, the L2CAP protocol specific information (PSM) points to the single instance of RFCOMM. In the last example, two different and independent RFCOMM instances are available on top of the L2CAP layer. In this case, the L2CAP protocol specific information (PSM) points to a distinct identifier that distinguishes each of the RFCOMM instances. See the L2CAP specification ([Vol. 3] Part A, Section 4.2) for the range of allowed PSM values.

*IrDA-like printer*

( ( L2CAP, PSM=RFCOMM ), ( RFCOMM, CN=1 ), ( PostscriptStream ) )

*IP Network Printing*

( ( L2CAP, PSM=RFCOMM ), ( RFCOMM, CN=2 ), ( PPP ), ( IP ), ( TCP ), ( IPP ) )

Synchronization Protocol Descriptor Example

( ( L2CAP, PSM=0x1001 ), ( RFCOMM, CN=1 ), ( Obex ), ( vCal ) )

( ( L2CAP, PSM=0x1003 ), ( RFCOMM, CN=1 ), ( Obex ), ( otherSynchronisationApplication ) )

### 5.1.6 AdditionalProtocolDescriptorList Attribute

| Attribute Name | Attribute ID | Attribute Value Type |
|---|---|---|
| AdditionalProtocolDescriptorList | 0x000D | Data Element Sequence |

**Description:**

The AdditionalProtocolDescriptorLists attribute contains a sequence of ProtocolDescriptorList-elements. Each element having the same format as the ProtocolDescriptorList described in section 5.1.5. The ordering of the elements is significant and should be specified and fixed in Profiles that make use of this attribute.

The AdditionalProtocolDescriptorLists attribute supports services that require more channels in addition to the service described in Section 5.1.5 . If the AdditionalProtocolDescriptorLists attribute is included in a service record, the ProtocolDescriptorList attribute must be included.

**AdditionalProtocolDescriptorList Examples**

The following is just an illustrative example and is not meant to refer a real specified service or protocols.

```
Attribute    Attribute Value type            Attribute Value

ProtocolDescriptorList

   ProtocolDescriptor #0DataElementSequence

         ProtocolID    UUID       L2CAP

         Param: PSM    PSM        FooDataProtocol

   ProtocolDescriptor #1DataElementSequence

          ProtocolID           UUID       FooDataProtocol

AdditionalProtocolDescriptorLists
```

```
ProtocolDescriptorList #0  DataElementSequence

ProtocolDescriptor #0      DataElementSequence

           ProtocolID      UUID       L2CAP

           Param: PSM      PSM        FooControlProtocol

      ProtocolDescriptor #1DataElementSequence

           ProtocolID      UUID       FooControlProtocol
```

### 5.1.7 BrowseGroupList Attribute

| Attribute Name | Attribute ID | Attribute Value Type |
|---|---|---|
| BrowseGroupList | 0x0005 | Data Element Sequence |

**Description:**

The BrowseGroupList attribute consists of a data element sequence in which each element is a UUID that represents a browse group to which the service record belongs. The top-level browse group ID, called PublicBrowseRoot and representing the root of the browsing hierarchy, has the value 00001002-0000-1000-8000-00805F9B34FB (UUID16: 0x1002) from the Bluetooth Assigned Numbers document.

### 5.1.8 LanguageBaseAttributeIDList Attribute

| Attribute Name | Attribute ID | Attribute Value Type |
|---|---|---|
| LanguageBaseAttributeIDList | 0x0006 | Data Element Sequence |

**Description:**

In order to support human-readable attributes for multiple natural languages in a single service record, a base attribute ID is assigned for each of the natural languages used in a service record. The human-readable universal attributes are then defined with an attribute ID offset from each of these base values, rather than with an absolute attribute ID.

The LanguageBaseAttributeIDList attribute is a list in which each member contains a language identifier, a character encoding identifier, and a base attribute ID for each of the natural languages used in the service record. The LanguageBaseAttributeIDList attribute consists of a data element sequence in which each element is a 16-bit unsigned integer. The elements are grouped as triplets (threes).

The first element of each triplet contains an identifier representing the natural language. The language is encoded according to ISO 639:1988 (E/F): “Code for the representation of names of languages”.

The second element of each triplet contains an identifier that specifies a character encoding used for the language. Values for character encoding can be found in IANA's database[1], and have the values that are referred to as MIBEnum values. The recommended character encoding is UTF-8.

The third element of each triplet contains an attribute ID that serves as the base attribute ID for the natural language in the service record. Different service records within a server may use different base attribute ID values for the same language.

To facilitate the retrieval of human-readable universal attributes in a principal language, the base attribute ID value for the primary language supported by a service record shall be 0x0100. Also, if a LanguageBaseAttributeIDList attribute is contained in a service record, the base attribute ID value contained in its first element shall be 0x0100. If one or more human readable attributes are contained in a service record, the LanguageBaseAttributeIDList attribute should be included in that service record.

### 5.1.9 ServiceInfoTimeToLive Attribute

| Attribute Name | Attribute ID | Attribute Value Type |
|---|---|---|
| ServiceInfoTimeToLive | 0x0007 | 32-bit unsigned integer |

**Description:**

The ServiceTimeToLive attribute is a 32-bit integer that contains the number of seconds for which the information in a service record is expected to remain valid and unchanged. This time interval is measured from the time that the attribute value is retrieved from the SDP server. This value does not imply a guarantee that the service record will remain available or unchanged. It is simply a hint that a client can use to determine a suitable polling interval to re-validate the service record contents.

1. See http://www.isi.edu/in-notes/iana/assignments/character-sets

### 5.1.10 ServiceAvailability Attribute

| Attribute Name | Attribute ID | Attribute Value Type |
|---|---|---|
| ServiceAvailability | 0x0008 | 8-bit unsigned integer |

**Description:**

The ServiceAvailability attribute is an 8-bit unsigned integer that represents the relative ability of the service to accept additional clients. A value of 0xFF indicates that the service is not currently in use and is thus fully available, while a value of 0x00 means that the service is not accepting new clients. For services that support multiple simultaneous clients, intermediate values indicate the relative availability of the service on a linear scale.

For example, a service that can accept up to 3 clients should provide ServiceAvailability values of 0xFF, 0xAA, 0x55, and 0x00 when 0, 1, 2, and 3 clients, respectively, are utilizing the service. The value 0xAA is approximately (2/3) * 0xFF and represents 2/3 availability, while the value 0x55 is approximately (1/3)*0xFF and represents 1/3 availability. Note that the availability value is be approximated as

( 1 - ( current_number_of_clients / maximum_number_of_clients ) ) * 0xFF

When the maximum number of clients is large, this formula must be modified to ensure that ServiceAvailability values of 0x00 and 0xFF are reserved for their defined meanings of unavailability and full availability, respectively.

Note that the maximum number of clients a service can support may vary according to the resources utilized by the service's current clients.

A non-zero value for ServiceAvailability does not guarantee that the service will be available for use. It should be treated as a hint or an approximation of availability status.

### 5.1.11 BluetoothProfileDescriptorList Attribute

| Attribute Name | Attribute ID | Attribute Value Type |
|---|---|---|
| BluetoothProfileDescriptorList | 0x0009 | Data Element Sequence |

**Description:**

The BluetoothProfileDescriptorList attribute consists of a data element sequence in which each element is a profile descriptor that contains information about a Bluetooth profile to which the service represented by this service record conforms. Each profile descriptor is a data element sequence whose first element is the UUID assigned to the profile and whose second element is a 16-bit profile version number.

Each version of a profile is assigned a 16-bit unsigned integer profile version number, which consists of two 8-bit fields. The higher-order 8 bits contain the

major version number field and the lower-order 8 bits contain the minor version number field. The initial version of each profile has a major version of 1 and a minor version of 0. When upward compatible changes are made to the profile, the minor version number will be incremented. If incompatible changes are made to the profile, the major version number will be incremented.

### 5.1.12 DocumentationURL Attribute

| Attribute Name | Attribute ID | Attribute Value Type |
|---|---|---|
| DocumentationURL | 0x000A | URL |

**Description:**

This attribute is a URL which points to documentation on the service described by a service record.

### 5.1.13 ClientExecutableURL Attribute

| Attribute Name | Attribute ID | Attribute Value Type |
|---|---|---|
| ClientExecutableURL | 0x000B | URL |

**Description:**

This attribute contains a URL that refers to the location of an application that can be used to utilize the service described by the service record. Since different operating environments require different executable formats, a mechanism has been defined to allow this single attribute to be used to locate an executable that is appropriate for the client device's operating environment. In the attribute value URL, the first byte with the value 0x2A (ASCII character '*') is to be replaced by the client application with a string representing the desired operating environment before the URL is to be used.

The list of standardized strings representing operating environments is contained in the Bluetooth Assigned Numbers document.

For example, assume that the value of the ClientExecutableURL attribute is http://my.fake/public/*/client.exe. On a device capable of executing SH3 WindowsCE files, this URL would be changed to http://my.fake/public/sh3-microsoft-wince/client.exe. On a device capable of executing Windows 98 binaries, this URL would be changed to http://my.fake/public/i86-microsoft-win98/client.exe.

### 5.1.14 IconURL Attribute

| Attribute Name | Attribute ID | Attribute Value Type |
|---|---|---|
| IconURL | 0x000C | URL |

**Description:**

This attribute contains a URL that refers to the location of an icon that can be used to represent the service described by the service record. Since different hardware devices require different icon formats, a mechanism has been defined to allow this single attribute to be used to locate an icon that is appropriate for the client device. In the attribute value URL, the first byte with the value 0x2A (ASCII character '*') is to be replaced by the client application with a string representing the desired icon format before the URL is to be used.

The list of standardized strings representing icon formats is contained in the Bluetooth Assigned Numbers document.

For example, assume that the value of the IconURL attribute is http://my.fake/public/icons/*. On a device that prefers 24 x 24 icons with 256 colors, this URL would be changed to http://my.fake/public/icons/24x24x8.png. On a device that prefers 10 x 10 monochrome icons, this URL would be changed to http://my.fake/public/icons/10x10x1.png.

### 5.1.15 ServiceName Attribute

| Attribute Name | Attribute ID Offset | Attribute Value Type |
|---|---|---|
| ServiceName | 0x0000 | String |

**Description:**

The ServiceName attribute is a string containing the name of the service represented by a service record. It should be brief and suitable for display with an Icon representing the service. The offset 0x0000 is added to the attribute ID base (contained in the LanguageBaseAttributeIDList attribute) in order to compute the attribute ID for this attribute.

### 5.1.16 ServiceDescription Attribute

| Attribute Name | Attribute ID Offset | Attribute Value Type |
|---|---|---|
| ServiceDescription | 0x0001 | String |

**Description:**

This attribute is a string containing a brief description of the service. It should be less than 200 characters in length. The offset 0x0001 is added to the attribute ID base (contained in the LanguageBaseAttributeIDList attribute) in order to compute the attribute ID for this attribute.

### 5.1.17 ProviderName Attribute

| Attribute Name | Attribute ID Offset | Attribute Value Type |
|---|---|---|
| ProviderName | 0x0002 | String |

**Description:**

This attribute is a string containing the name of the person or organization providing the service. The offset 0x0002 is added to the attribute ID base (contained in the LanguageBaseAttributeIDList attribute) in order to compute the attribute ID for this attribute.

### 5.1.18 Reserved Universal Attribute IDs

Attribute IDs in the range of 0x000E – 0x01FF are reserved for use by SDP, if allocated the assigned numbers document will be updated.

## 5.2 SERVICEDISCOVERYSERVER SERVICE CLASS ATTRIBUTE DEFINITIONS

This service class describes service record that contains attributes of service discovery server itself. The attributes listed in this section are only valid if the ServiceClassIDList attribute contains the ServiceDiscoveryServerServiceClassID. All of the universal attributes may be included in service records of the ServiceDiscoveryServer class.

### 5.2.1 ServiceRecordHandle Attribute

Described in the universal attribute definition for ServiceRecordHandle.

**Value**

A 32-bit integer with the value 0x000000000.

### 5.2.2 ServiceClassIDList Attribute

Described in the universal attribute definition for ServiceClassIDList.

**Value**

A UUID representing the ServiceDiscoveryServerServiceClassID.

### 5.2.3 VersionNumberList Attribute

| Attribute Name | Attribute ID | Attribute Value Type |
|---|---|---|
| VersionNumberList | 0x0200 | Data Element Sequence |

**Description:**

The VersionNumberList is a data element sequence in which each element of the sequence is a version number supported by the SDP server.

A version number is a 16-bit unsigned integer consisting of two fields. The higher-order 8 bits contain the major version number field and the low-order 8 bits contain the minor version number field. The initial version of SDP has a major version of 1 and a minor version of 0. When upward compatible changes are made to the protocol, the minor version number will be incremented. If incompatible changes are made to SDP, the major version number will be incremented. This guarantees that if a client and a server support a common major version number, they can communicate if each uses only features of the specification with a minor version number that is supported by both client and server.

### 5.2.4 ServiceDatabaseState Attribute

| Attribute Name | Attribute ID | Attribute Value Type |
|---|---|---|
| ServiceDatabaseState | 0x0201 | 32-bit unsigned integer |

**Description:**

The ServiceDatabaseState is a 32-bit integer that is used to facilitate caching of service records. If this attribute exists, its value shall be changed when any of the other service records are added to or deleted from the server's database. If this value has not changed since the last time a client queried its value, the client knows that a) none of the other service records maintained by the SDP server have been added or deleted; and b) any service record handles acquired from the server are still valid. A client shall query this attribute's value when a connection to the server is established, prior to using any service record handles acquired during a previous connection.

Note that the ServiceDatabaseState attribute does not change when existing service records are modified, including the addition, removal, or modification of service attributes. A service record's ServiceRecordState attribute indicates when that service record is modified.

### 5.2.5 Reserved Attribute IDs

Attribute IDs in the range of 0x0202-0x02FF are reserved.

## 5.3 BROWSEGROUPDESCRIPTOR SERVICE CLASS ATTRIBUTE DEFINITIONS

This service class describes the ServiceRecord provided for each BrowseGroupDescriptor service offered on a Bluetooth device. The attributes listed in this section are only valid if the ServiceClassIDList attribute contains the BrowseGroupDescriptorServiceClassID. Note that all of the universal attributes may be included in service records of the BrowseGroupDescriptor class.

### 5.3.1 ServiceClassIDList Attribute

Described in the universal attribute definition for ServiceClassIDList.

**Value**

A UUID representing the BrowseGroupDescriptorServiceClassID.

### 5.3.2 GroupID Attribute

| Attribute Name | Attribute ID | Attribute Value Type |
|---|---|---|
| GroupID | 0x0200 | UUID |

**Description:**

This attribute contains a UUID that can be used to locate services that are members of the browse group that this service record describes.

### 5.3.3 Reserved Attribute IDs

Attribute IDs in the range of 0x0201-0x02FF are reserved.

# 6 SECURITY

In Security Mode 4, SDP should use no security but may use security (an authenticated or unauthenticated link key with encryption). See [Part C] Section 5.2.2 on page 299).

# LIST OF FIGURES

Figure 2.1: SDP Client-Server Interaction ..... 211
Figure 2.2: Simplified SDP Client-Server Interaction ..... 211
Figure 2.3: Service Record ..... 213
Figure 2.4: Service Attribute ..... 214
Figure 2.5: Service Browsing Hierarchy ..... 218
Figure 3.1: Data Element ..... 222
Figure 4.1: Protocol Data Unit Format ..... 223
Figure 4.2: Continuation State Format ..... 225
Figure 4.3: Error Handling ..... 226
Figure 4.4: ServiceSearch Transaction ..... 227
Figure 4.5: ServiceAttribute Transaction ..... 230
Figure 4.6: ServiceSearchAttribute Transaction ..... 233

# LIST OF TABLES

Table 2.1: Service Browsing Hierarchy ..... 219
Table 3.1: Data Element ..... 220
Table 3.2: Data Element Size ..... 221

# APPENDIX A – BACKGROUND INFORMATION

## A.1. Service Discovery

As computing continues to move to a network-centric model, finding and making use of services that may be available in the network becomes increasingly important. Services can include common ones such as printing, paging, FAXing, and so on, as well as various kinds of information access such as teleconferencing, network bridges and access points, eCommerce facilities, and so on — most any kind of service that a server or service provider might offer. In addition to the need for a standard way of discovering available services, there are other considerations: getting access to the services (finding and obtaining the protocols, access methods, “drivers” and other code necessary to utilize the service), controlling access to the services, advertising the services, choosing among competing services, billing for services, and so on. This problem is widely recognized; many companies, standards bodies and consortia are addressing it at various levels in various ways. Service Location Protocol (SLP), Jini™, and Salutation™, to name just a few, all address some aspect of service discovery.

## A.2. Bluetooth Service Discovery

Bluetooth Service Discovery Protocol (SDP) addresses service discovery specifically for the Bluetooth environment. It is optimized for the highly dynamic nature of Bluetooth communications. SDP focuses primarily on discovering services available from or through Bluetooth devices. SDP does not define methods for accessing services; once services are discovered with SDP, they can be accessed in various ways, depending upon the service. This might include the use of other service discovery and access mechanisms such as those mentioned above; SDP provides a means for other protocols to be used along with SDP in those environments where this can be beneficial. While SDP can coexist with other service discovery protocols, it does not require them. In Bluetooth environments, services can be discovered using SDP and can be accessed using other protocols defined by Bluetooth.

# APPENDIX B – EXAMPLE SDP TRANSACTIONS

The following are simple examples of typical SDP transactions. These are meant to be illustrative of SDP flows. The examples do not consider:

- Caching (in a caching system, the SDP client would make use of the ServiceRecordState and ServiceDatabaseState attributes);
- Service availability (if this is of interest, the SDP client should use the ServiceAvailability and/or ServiceTimeToLive attributes);
- SDP versions (the VersionNumberList attribute could be used to determine compatible SDP versions);
- SDP Error Responses (an SDP error response is possible for any SDP request that is in error); and
- Communication connection (the examples assume that an L2CAP connection is established).

The examples are meant to be illustrative of the protocol. The format used is `ObjectName[ObjectSizeInBytes] {SubObjectDefinitions},` but this is not meant to illustrate an interface. The `ObjectSizeInBytes` is the size of the object in decimal. The `SubObjectDefinitions` (inside of `{}` characters) are components of the immediately enclosing object. Hexadecimal values shown as lower-case letters, such as for transaction IDs and service handles, are variables (the particular value is not important for the illustration, but each such symbol always represents the same value). Comments are included in this manner: `/* comment text */`.

Numeric values preceded by “0x” are hexadecimal, while those preceded by “0b” are binary. All other numeric values are decimal.

## B.1. SDP Example 1 – ServiceSearchRequest

The first example is that of an SDP client searching for a generic printing service. The client does not specify a particular type of printing service. In the example, the SDP server has two available printing services. The transaction illustrates:

1. SDP client to SDP server: SDP_ServiceSearchRequest, specifying the PrinterServiceClassID (represented as a DataElement with a 32-bit UUID value of `ppp...ppp`) as the only element of the ServiceSearchPattern. The PrinterServiceClassID is assumed to be a 32-bit UUID and the data element type for it is illustrated. The TransactionID is illustrated as `tttt`.
2. SDP server to SDP client: SDP_ServiceSearchResponse, returning handles to two printing services, represented as `qqqqqqqq` for the first printing service and `rrrrrrrr` for the second printing service. The Transaction ID is the same value as supplied by the SDP client in the corresponding request (`tttt`).

```
/* Sent from SDP Client to SDP server */
SDP_ServiceSearchRequest[15] {
  PDUID[1] {
    0x02
  }
  TransactionID[2] {
    0xtttt
  }
  ParameterLength[2] {
    0x000A
  }
  ServiceSearchPattern[7] {
    DataElementSequence[7] {
      0b00110 0b101 0x05
      UUID[5] {
        /* PrinterServiceClassID */
        0b00011 0b010 0xpppppppp
      }
    }
  }
  MaximumServiceRecordCount[2] {
    0x0003
  }
  ContinuationState[1] {
    /* no continuation state */
    0x00
  }
}


/* Sent from SDP server to SDP client */
SDP_ServiceSearchResponse[18] {
  PDUID[1] {
    0x03
  }
  TransactionID[2] {
    0xtttt
  }
  ParameterLength[2] {
    0x000D
  }
  TotalServiceRecordCount[2] {
    0x0002
  }
  CurrentServiceRecordCount[2] {
    0x0002
  }
  ServiceRecordHandleList[8] {
    /* print service 1 handle */
    0xqqqqqqqq
    /* print service 2 handle */
    0xrrrrrrrr
  }
  ContinuationState[1] {
    /* no continuation state */
```

```
    0x00
  }
}
```

## B.2. SDP Example 2 – ServiceAttributeTransaction

The second example continues the first example. In Example 1, the SDP client obtained handles to two printing services. In Example 2, the client uses one of those service handles to obtain the ProtocolDescriptorList attribute for that printing service. The transaction illustrates:

1. SDP client to SDP server: SDP_ServiceAttributeRequest, presenting the previously obtained service handle (the one denoted as qqqqqqqq) and specifying the ProtocolDescriptorList attribute ID (AttributeID 0x0004) as the only attribute requested (other attributes could be retrieved in the same transaction if desired). The TransactionID is illustrated as uuuu to distinguish it from the TransactionID of Example 1.
2. SDP server to SDP client: SDP_ServiceAttributeResponse, returning the ProtocolDescriptorList for the specified printing service. This protocol stack is assumed to be ( (L2CAP), (RFCOMM, 2), (PostscriptStream) ). The ProtocolDescriptorList is a data element sequence in which each element is, in turn, a data element sequence whose first element is a UUID representing the protocol, and whose subsequent elements are protocol-specific parameters. In this example, one such parameter is included for the RFCOMM protocol, an 8-bit value indicating RFCOMM server channel 2. The Transaction ID is the same value as supplied by the SDP client in the corresponding request (uuuu). The Attributes returned are illustrated as a data element sequence where the protocol descriptors are 32-bit UUIDs and the RFCOMM server channel is a data element with an 8-bit value of 2.

```
 /* Sent from SDP Client to SDP server */
SDP_ServiceAttributeRequest[17] {
  PDUID[1] {
    0x04
  }
  TransactionID[2] {
    0xuuuu
  }
  ParameterLength[2] {
    0x000C
  }
  ServiceRecordHandle[4] {
    0xqqqqqqqq
  }
  MaximumAttributeByteCount[2] {
    0x0080
  }
  AttributeIDList[5] {
    DataElementSequence[5] {
```

```
      0b00110 0b101 0x03
      AttributeID[3] {
        0b00001 0b001 0x0004
      }
    }
  }
  ContinuationState[1] {
    /* no continuation state */
    0x00
  }
}

/* Sent from SDP server to SDP client */
SDP_ServiceAttributeResponse[38] {
  PDUID[1] {
    0x05
  }
  TransactionID[2] {
    0xuuuu
  }
  ParameterLength[2] {
    0x0021
  }
  AttributeListByteCount[2] {
    0x001E
  }
  AttributeList[30] {
    DataElementSequence[30] {
      0b00110 0b101 0x1C
      Attribute[28] {
        AttributeID[3] {
          0b00001 0b001 0x0004
        }
        AttributeValue[25] {
          /* ProtocolDescriptorList */
          DataElementSequence[25] {
            0b00110 0b101 0x17
            /* L2CAP protocol descriptor */
            DataElementSequence[7] {
              0b00110 0b101 0x05
              UUID[5] {
                /* L2CAP Protocol UUID */
                0b00011 0b010 <32-bit L2CAP UUID>
              }
            }
            /* RFCOMM protocol descriptor */
            DataElementSequence[9] {
              0b00110 0b101 0x07
              UUID[5] {
                /* RFCOMM Protocol UUID */
                0b00011 0b010 <32-bit RFCOMM UUID>
              }
              /* parameter for server 2 */
              Uint8[2] {
                0b00001 0b000 0x02
              }
            }
```

```
            /* PostscriptStream protocol descriptor */
            DataElementSequence[7] {
              0b00110 0b101 0x05
              UUID[5] {
                /* PostscriptStream Protocol UUID */
                0b00011 0b010 <32-bit PostscriptStream UUID>
              }
            }
          }
        }
      }
    }
  }
  ContinuationState[1] {
    /* no continuation state */
    0x00
  }
}
```

## B.3. SDP Example 3 – ServiceSearchAttributeTransaction

The third example is a form of service browsing, although it is not generic browsing in that it does not make use of SDP browse groups. Instead, an SDP client is searching for available Synchronization services that can be presented to the user for selection. The SDP client does not specify a particular type of synchronization service. In the example, the SDP server has three available synchronization services: an address book synchronization service and a calendar synchronization service (both from the same provider), and a second calendar synchronization service from a different provider. The SDP client is retrieving the same attributes for each of these services; namely, the data formats supported for the synchronization service (vCard, vCal, ICal, etc.) and those attributes that are relevant for presenting information to the user about the services. Also assume that the maximum size of a response is 400 bytes. Since the result is larger than this, the SDP client will repeat the request supplying a continuation state parameter to retrieve the remainder of the response. The transaction illustrates:

1. SDP client to SDP server: SDP_ServiceSearchAttributeRequest, specifying the generic SynchronisationServiceClassID (represented as a data element whose 32-bit UUID value is `sss...sss`) as the only element of the ServiceSearchPattern. The SynchronisationServiceClassID is assumed to be a 32-bit UUID. The requested attributes are the ServiceRecordHandle (AttributeID 0x0000), ServiceClassIDList (AttributeID 0x0001), IconURL (AttributeID 0x000C), ServiceName (AttributeID 0x0100), ServiceDescription (AttributeID 0x0101), and ProviderName (AttributeID 0x0102) attributes; as well as the service-specific SupportedDataStores (AttributeID 0x0301). Since the first two attribute IDs (0x0000 and 0x0001) and three other attribute IDs(0x0100, 0x0101, and 0x0102 are consecutive, they are specified as attribute ranges. The TransactionID is illustrated as `vvvv` to distinguish it from the TransactionIDs of the other Examples.

Note that values in the service record's primary language are requested for the text attributes (ServiceName, ServiceDescription and ProviderName) so that absolute attribute IDs may be used, rather than adding offsets to a base obtained from the LanguageBaseAttributeIDList attribute.

2. SDP server to SDP client: SDP_ServiceSearchAttributeResponse, returning the specified attributes for each of the three synchronization services. In the example, each ServiceClassIDList is assumed to contain a single element, the generic SynchronisationServiceClassID (a 32-bit UUID represented as sss...sss). Each of the other attributes contain illustrative data in the example (the strings have illustrative text; the icon URLs are illustrative, for each of the respective three synchronization services; and the SupportedDataStore attribute is represented as an unsigned 8-bit integer where 0x01 = vCard2.1, 0x02 = vCard3.0, 0x03 = vCal1.0 and 0x04 = iCal). Note that one of the service records (the third for which data is returned) has no ServiceDescription attribute. The attributes are returned as a data element sequence, where each element is in turn a data element sequence representing a list of attributes. Within each attribute list, the ServiceClassIDList is a data element sequence while the remaining attributes are single data elements. The Transaction ID is the same value as supplied by the SDP client in the corresponding request (0xvvvv). Since the entire result cannot be returned in a single response, a non-null continuation state is returned in this first response.
3. Note that the total length of the initial data element sequence (487 in the example) is indicated in the first response, even though only a portion of this data element sequence (368 bytes in the example, as indicated in the AttributeLists byte count) is returned in the first response. The remainder of this data element sequence is returned in the second response (without an additional data element header).
4. SDP client to SDP server: SDP_ServiceSearchAttributeRequest, with the same parameters as in step 1, except that the continuation state received from the server in step 2 is included as a request parameter. The TransactionID is changed to 0xwww to distinguish it from previous request.
5. SDP server to SDP client: SDP_ServiceSearchAttributeResponse, with the remainder of the result computed in step 2 above. Since all of the remaining result fits in this second response, a null continuation state is included.

```
/* Part 1 -- Sent from SDP Client to SDP server */
SdpSDP_ServiceSearchAttributeRequest[33] {
  PDUID[1] {
    0x06
  }
  TransactionID[2] {
    0xvvvv
  }
  ParameterLength[2] {
    0x001C
  }
  ServiceSearchPattern[7] {
    DataElementSequence[7] {
```

```
      0b00110 0b101 0x05
      UUID[5] {
        /* SynchronisationServiceClassID */
        0b00011 0b010 0xssssssss
      }
    }
  }
  MaximumAttributeByteCount[2] {
    0x0190
  }
AttributeIDList[18] {
    DataElementSequence[18] {
      0b00110 0b101 0x10
      AttributeIDRange[5] {
        0b00001 0b010 0x00000001
      }
      AttributeID[3] {
        0b00001 0b001 0x000C
      }
      AttributeIDRange[5] {
        0b00001 0b010 0x01000102
      }
      AttributeID[3] {
        0b00001 0b001 0x0301
      }    }
  }
  ContinuationState[1] {
    /* no continuation state */
    0x00
  }
}

/* Part 2 -- Sent from SDP server to SDP client */
SdpSDP_ServiceSearchAttributeResponse[384] {
  PDUID[1] {
    0x07
  }
  TransactionID[2] {
    0xvvvv
  }
  ParameterLength[2] {
    0x017B
  }
  AttributeListByteCount[2] {
    0x0170
  }
  AttributeLists[368] {
    DataElementSequence[487] {
      0b00110 0b110 0x01E4
      DataElementSequence[178] {
        0b00110 0b101 0xB0
        Attribute[8] {
          AttributeID[3] {
            0b00001 0b001 0x0000
          }
          AttributeValue[5] {
            /* service record handle */
```

```
            0b00001 0b010 0xhhhhhhhh
          }
        }
        Attribute[10] {
          AttributeID[3] {
            0b00001 0b001 0x0001
          }
AttributeValue[7] {
            DataElementSequence[7] {
              0b00110 0b101 0x05
              UUID[5] {
                /* SynchronisationServiceClassID */
                0b00011 0b010 0xssssssss
              }
            }
          }
        }
        Attribute[35] {
          AttributeID[3] {
            0b00001 0b001 0x000C
          }
          AttributeValue[32] {
            /* IconURL; '*' replaced by client application */
            0b01000 0b101 0x1E
            "http://Synchronisation/icons/*"
          }
        }
        Attribute[22] {
          AttributeID[3] {
            0b00001 0b001 0x0100
          }
          AttributeValue[19] {
            /* service name */
            0b00100 0b101 0x11
            "Address Book Sync"
          }
        }
        Attribute[59] {
          AttributeID[3] {
            0b00001 0b001 0x0101
          }
          AttributeValue[56] {
            /* service description */
            0b00100 0b101 0x36
            "Synchronisation Service for"
            " vCard Address Book Entries"
          }
        }
        Attribute[37] {
          AttributeID[3] {
            0b00001 0b001 0x0102
          }
          AttributeValue[34] {
            /* service provider */
            0b00100 0b101 0x20
            "Synchronisation Specialists Inc."
          }
```

```
        }
        Attribute[5] {
          AttributeID[3] {
            0b00001 0b001 0x0301
          }
          AttributeValue[2] {
            /* Supported Data Store 'phonebook' */
            0b00001 0b000 0x01
          }
        }
      }
      DataElementSequence[175] {
        0b00110 0b101 0xAD
        Attribute[8] {
          AttributeID[3] {
            0b00001 0b001 0x0000
          }
          AttributeValue[5] {
            /* service record handle */
            0b00001 0b010 0xmmmmmmmm
          }
        }
        Attribute[10] {
          AttributeID[3] {
            0b00001 0b001 0x0001
          }
                    AttributeValue[7] {
          DataElementSequence[7] {
            0b00110 0b101 0x05
            UUID[5] {
              /* SynchronisationServiceClassID */
              0b00011 0b010 0xssssssss
            }
          }
        }
        Attribute[35] {
          AttributeID[3] {
            0b00001 0b001 0x000C
          }
          AttributeValue[32] {
            /* IconURL; '*' replaced by client application */
            0b01000 0b101 0x1E
            "http://Synchronisation/icons/*"
          }
        }
        Attribute[21] {
          AttributeID[3] {
            0b00001 0b001 0x0100
          }
          AttributeValue[18] {
            /* service name */
            0b00100 0b101 0x10
            "Appointment Sync"
          }
        }
        Attribute[57] {
```

```
          AttributeID[3] {
            0b00001 0b001 0x0101
          }
          AttributeValue[54] {
            /* service description */
            0b00100 0b101 0x34
            "Synchronisation Service for"
            " vCal Appointment Entries"
          }
        }
        Attribute[37] {
          AttributeID[3] {
            0b00001 0b001 0x0102
          }
          AttributeValue[34] {
            /* service provider */
            0b00100 0b101 0x20
            "Synchronisation Specialists Inc."
          }
        }
        Attribute[5] {
          AttributeID[3] {
            0b00001 0b001 0x0301
          }
          AttributeValue[2] {
            /* Supported Data Store 'calendar' */
            0b00001 0b000 0x03
          }
        }
      }
    /* } Data element sequence of attribute lists */
    /* is not completed in this PDU. */
  }
  ContinuationState[9] {
    /* 8 bytes of continuation state */
    0x08 0xzzzzzzzzzzzzzzzz
  }
}

/* Part 3 -- Sent from SDP Client to SDP server */
SdpSDP_ServiceSearchAttributeRequest[41] {
  PDUID[1] {
    0x06
  }
  TransactionID[2] {
    0xwwww
  }
  ParameterLength[2] {
    0x0024
  }
  ServiceSearchPattern[7] {
    DataElementSequence[7] {
      0b00110 0b101 0x05
      UUID[5] {
        /* SynchronisationServiceClassID */
        0b00011 0b010 0xssssssss
      }
```

```
    }
  }
  MaximumAttributeByteCount[2] {
    0x0180
  }
    AttributeIDList[18] {
    DataElementSequence[18] {
      0b00110 0b101 0x10
      AttributeIDRange[5] {
        0b00001 0b010 0x00000001
      }
      AttributeID[3] {
        0b00001 0b001 0x000C
      }
      AttributeIDRange[5] {
        0b00001 0b010 0x01000102
      }
      AttributeID[3] {
        0b00001 0b001 0x0301
      }
    }
  }
  ContinuationState[9] {
    /* same 8 bytes of continuation state */
    /* received in part 2 */
    0x08 0xzzzzzzzzzzzzzzzz
  }
}

Part 4 -- Sent from SDP server to SDP client

SdpSDP_ServiceSearchAttributeResponse[115] {
  PDUID[1] {
    0x07
  }
  TransactionID[2] {
    0xwwww
  }
  ParameterLength[2] {
    0x006E
  }
  AttributeListByteCount[2] {
    0x006B
  }
  AttributeLists[107] {
    /* Continuing the data element sequence of */
    /* attribute lists begun in Part 2. */
      DataElementSequence[107] {
        0b00110 0b101 0x69
        Attribute[8] {
          AttributeID[3] {
            0b00001 0b001 0x0000
          }
          AttributeValue[5] {
            /* service record handle */
            0b00001 0b010 0xffffffff
          }
```

```
        }
        Attribute[10] {
          AttributeID[3] {
            0b00001 0b001 0x0001
          }
          AttributeValue[7] {
            DataElementSequence[7] {
              0b00110 0b101 0x05
              UUID[5] {
                /* SynchronisationServiceClassID */
                0b00011 0b010 0xssssssss
              }
            }
          }
        }
        Attribute[35] {
          AttributeID[3] {
            0b00001 0b001 0x000C
          }
          AttributeValue[32] {
            /* IconURL; '*' replaced by client application */
            0b01000 0b101 0x1E
            "http://DevManufacturer/icons/*"
          }
        }
        Attribute[18] {
          AttributeID[3] {
            0b00001 0b001 0x0100
          }
          AttributeValue[15] {
            /* service name */
            0b00100 0b101 0x0D
            "Calendar Sync"
          }
        }
        Attribute[29] {
          AttributeID[3] {
            0b00001 0b001 0x0102
          }
          AttributeValue[26] {
            /* service provider */
            0b00100 0b101 0x18
            "Device Manufacturer Inc."
          }
        }
        Attribute[5] {
          AttributeID[3] {
            0b00001 0b001 0x0301
          }
          AttributeValue[2] {
            /* Supported Data Store 'calendar' */
            0b00001 0b000 0x03
          }
        }
      }
    /* This completes the data element sequence */
    /* of attribute lists begun in Part 2.
```

# GENERIC ACCESS PROFILE

***This profile defines the generic procedures related to discovery of Bluetooth devices (idle mode procedures) and link management aspects of connecting to Bluetooth devices (connecting mode procedures). It also defines procedures related to use of different security levels. In addition, this profile includes common format requirements for parameters accessible on the user interface level.***


Ford Ex. 1052

# CONTENTS

**1 Introduction ..... 274**
1.1 Scope ..... 274
1.2 Symbols and Conventions ..... 275
1.2.1 Requirement Status Symbols ..... 275
1.2.2 Signaling diagram conventions ..... 276
1.2.3 Notation for Timers and Counters ..... 276
**2 Profile Overview ..... 277**
2.1 Profile Stack ..... 277
2.2 Profile Roles ..... 277
2.2.1 Roles when Operating over BR/EDR Physical Channel ..... 277
2.2.2 Roles when Operating over an LE Physical Channel ..... 278
2.3 User Requirements and Scenarios ..... 280
2.4 Profile Fundamentals ..... 281
2.5 Conformance ..... 281
2.6 Other Requirements ..... 281
**3 User Interface Aspects ..... 282**
3.1 The User Interface Level ..... 282
3.2 Representation of Bluetooth Parameters ..... 282
3.2.1 Bluetooth Device Address (BD_ADDR) ..... 282
3.2.2 Bluetooth Device Name (the user-friendly name) ..... 283
3.2.3 Bluetooth Passkey (Bluetooth PIN) ..... 284
3.2.4 Class of Device ..... 286
3.3 Pairing ..... 286
**4 Modes ..... 287**
4.1 Discoverability Modes ..... 287
4.1.1 Non-discoverable Mode ..... 288
4.1.2 Limited Discoverable Mode ..... 288
4.1.3 General Discoverable Mode ..... 289
4.2 Connectability Modes ..... 291
4.2.1 Non-connectable Mode ..... 291
4.2.2 Connectable Mode ..... 291
4.3 Bondable Modes ..... 293
4.3.1 Non-bondable Mode ..... 293
4.3.2 Bondable Mode ..... 293
**5 Security Aspects ..... 295**
5.1 Authentication ..... 295

5.1.1 Purpose ..... 295
5.1.2 Term on UI level ..... 295
5.1.3 Procedure ..... 296
5.1.4 Conditions ..... 296
5.2 Security Modes ..... 296
5.2.1 Legacy Security Modes ..... 297
5.2.2 Security Mode 4 (service level enforced security) ..... 299
**6 Idle Mode Procedures ..... 314**
6.1 General Inquiry ..... 314
6.1.1 Purpose ..... 314
6.1.2 Term on UI level ..... 314
6.1.3 Description ..... 315
6.1.4 Conditions ..... 315
6.2 Limited Inquiry ..... 315
6.2.1 Purpose ..... 315
6.2.2 Term on UI level ..... 316
6.2.3 Description ..... 316
6.2.4 Conditions ..... 316
6.3 Name Discovery ..... 317
6.3.1 Purpose ..... 317
6.3.2 Term on UI level ..... 317
6.3.3 Description ..... 317
6.3.4 Conditions ..... 318
6.4 Device Discovery ..... 318
6.4.1 Purpose ..... 318
6.4.2 Term on UI Level ..... 318
6.4.3 Description ..... 319
6.4.4 Conditions ..... 319
6.5 Bonding ..... 319
6.5.1 Purpose ..... 319
6.5.2 Term on UI level ..... 319
6.5.3 Description ..... 320
6.5.4 Conditions ..... 321
**7 Establishment Procedures ..... 323**
7.1 Link Establishment ..... 323
7.1.1 Purpose ..... 323
7.1.2 Term on UI Level ..... 323
7.1.3 Description ..... 323

7.1.4 Conditions ....325
7.2 Channel Establishment ....326
7.2.1 Purpose ....326
7.2.2 Term on UI level ....326
7.2.3 Description ....326
7.2.4 Conditions ....327
7.3 Connection Establishment ....328
7.3.1 Purpose ....328
7.3.2 Term on UI level ....328
7.3.3 Description ....328
7.3.4 Conditions ....329
7.4 Establishment of Additional Connection ....329
**8 Extended Inquiry Response Data Format ....330**
8.1 EIR Data Type Definitions ....331
8.1.1 Service Class UUIDs ....331
8.1.2 Local Name ....332
8.1.3 Flags ....332
8.1.4 Manufacturer Specific Data ....332
8.1.5 TX Power Level ....333
8.1.6 Secure Simple Pairing Out of Band (OOB) ....333
8.2 Example Extended Inquiry Response ....333
**9 Operational Modes and Procedures For Use On LE Physical Channels ....335**
9.1 Broadcast Mode and Observation Procedure ....335
9.1.1 Broadcast Mode ....335
9.1.2 Observation Procedure ....336
9.2 Discovery Modes and Procedures ....337
9.2.1 Requirements ....338
9.2.2 Non-Discoverable Mode ....338
9.2.3 Limited Discoverable Mode ....338
9.2.4 General Discoverable Mode ....340
9.2.5 Limited Discovery Procedure ....341
9.2.6 General Discovery Procedure ....343
9.2.7 Name Discovery Procedure ....344
9.3 Connection Modes and Procedures ....344
9.3.1 Requirements ....346
9.3.2 Non-Connectable Mode ....346
9.3.3 Directed Connectable Mode ....347
9.3.4 Undirected Connectable Mode ....349

9.3.5 Auto Connection Establishment Procedure ..... 351
9.3.6 General Connection Establishment Procedure ..... 354
9.3.7 Selective Connection Establishment Procedure ..... 356
9.3.8 Direct Connection Establishment Procedure ..... 357
9.3.9 Connection Parameter Update Procedure ..... 358
9.3.10 Terminate Connection Procedure ..... 358
9.4 Bonding Modes and Procedures ..... 359
9.4.1 Requirements ..... 359
9.4.2 Non-Bondable Mode ..... 359
9.4.3 Bondable Mode ..... 360
9.4.4 Bonding Procedure ..... 360
**10 LE Security Aspects ..... 362**
10.1 Requirements ..... 362
10.2 LE Security Modes ..... 362
10.2.1 LE Security Mode 1 ..... 362
10.2.2 LE Security Mode 2 ..... 363
10.2.3 Mixed Security Modes Requirements ..... 363
10.3 Authentication Procedure ..... 363
10.3.1 Receiving a Service Request ..... 364
10.3.2 Initiating a Service Request ..... 365
10.4 Data Signing ..... 368
10.4.1 Connection Data Signing Procedure ..... 368
10.4.2 Authenticate Signed Data Procedure ..... 368
10.5 Authorization Procedure ..... 369
10.6 Encryption Procedure ..... 369
10.7 Privacy Feature ..... 369
10.7.1 Privacy Feature in a Peripheral ..... 370
10.7.2 Privacy Feature in a Central ..... 371
10.8 Random Device Address ..... 371
10.8.1 Static Address ..... 371
10.8.2 Private address ..... 372
**11 Advertising and Scan Response Data Format ..... 375**
11.1 AD Type Definitions ..... 375
11.1.1 Service UUIDs ..... 376
11.1.2 Local Name ..... 376
11.1.3 Flags ..... 376
11.1.4 Manufacturer Specific Data ..... 376
11.1.5 TX Power Level ..... 377
11.1.6 Security Manager Out of Band (OOB) ..... 377

11.1.7 Security Manager TK Value ..... 377
11.1.8 Slave Connection Interval Range ..... 377
11.1.9 Service Solicitation ..... 377
11.1.10 Service Data ..... 377
11.2 Example Advertising Data ..... 378

**12 GAP Characteristics for Low Energy ..... 379**
12.1 Device Name Characteristic ..... 379
12.2 Appearance Characteristic ..... 380
12.3 Peripheral Privacy Flag Characteristic ..... 380
12.4 Reconnection Address Characteristic ..... 380
12.5 Peripheral Preferred Connection Parameters Characteristic ..... 381

**13 BR/EDR/LE Operation Modes and Procedure ..... 383**
13.1 Modes ..... 383
13.1.1 Discoverability ..... 384
13.1.2 Connectability ..... 385
13.1.3 Bondable Modes ..... 385
13.2 Idle Mode Procedures ..... 385
13.2.1 General Discovery Procedure ..... 386
13.2.2 Limited Discovery Procedure ..... 386
13.2.3 Device Discovery Procedure ..... 387
13.2.4 Name Discovery ..... 387
13.3 Establishment Procedures ..... 388
13.3.1 Link Establishment ..... 388
13.4 SDP Interoperability Requirements ..... 388

**14 BR/EDR/LE Security Aspects ..... 390**

**15 Definitions ..... 391**
15.1 General Definitions ..... 391
15.2 Connection-related Definitions ..... 391
15.3 Device-related Definitions ..... 392
15.4 Procedure-related Definitions ..... 393
15.5 Security-related Definitions ..... 393

**16 Appendix A (Normative): Timers and Constants ..... 395**

**17 Appendix B (Informative): Information Flows of Related Procedures ..... 398**
17.1 LMP – Authentication ..... 398
17.2 LMP – Pairing ..... 399
17.3 Service Discovery ..... 400
17.4 Generating a Resolvable Private Address ..... 400
17.5 Resolving a Resolvable Private Address ..... 400

**18 Appendix C (Normative): EIR and AD Formats ..... 401**

18.1 Flags ..... 401
18.2 Service ..... 402
18.3 Local Name ..... 402
18.4 TX Power Level ..... 402
18.5 Simple Pairing Optional OOB Tags ..... 403
18.6 Security Manager TK Value ..... 403
18.7 Security Manager OOB Flags ..... 403
18.8 Slave Connection Interval Range ..... 404
18.9 Service Solicitation ..... 404
18.10 Service Data ..... 404
18.11 Manufacturer Specific Data ..... 405

**19 References ..... 406**

# FOREWORD

Interoperability between devices from different manufacturers is provided for a specific service and use case, if the devices conform to a Bluetooth SIG-defined profile specification. A profile defines a selection of messages and procedures (generally termed *capabilities*) from the Bluetooth SIG specifications and gives a description of the air interface for specified service(s) and use case(s).

All defined features are process-mandatory. This means that, if a feature is used, it is used in a specified manner. Whether the provision of a feature is mandatory or optional is stated separately for both sides of the Bluetooth air interface.

# 1 INTRODUCTION

## 1.1 SCOPE

The purpose of the Generic Access Profile is:

To introduce definitions, recommendations and common requirements related to modes and access procedures that are to be used by transport and application profiles.

To describe how devices are to behave in standby and connecting states in order to guarantee that links and channels always can be established between Bluetooth devices, and that multi-profile operation is possible. Special focus is put on discovery, link establishment and security procedures.

To state requirements on user interface aspects, mainly coding schemes and names of procedures and parameters, that are needed to guarantee a satisfactory user experience.

This profile defines three device types based on the supported Core Configurations as defined in [Vol. 0], Part B Section 3.1. The device types are shown in Table 1.1:

| Device Type | Description |
|---|---|
| BR/EDR | Devices that support the “Basic Rate” Core Configuration (see [Vol. 0], Part B Section 4.1) |
| LE only | Devices that support the “Low Energy” Core Configuration (see [Vol. 0], Part B Section 4.4) |
| BR/EDR/LE | Devices that support the “Basic Rate and Low Energy Combined” Core Configuration (see [Vol. 0], Part B Section 4.5) |

*Table 1.1: Device Types*

Devices of the LE-only and BR/EDR/LE device types are capable of operating over an LE physical channel.

The terms physical links and physical channels as defined in [Vol. 1], Part A Section 3.3 and Section 4 are used in this specification.

The requirements for device types are shown in Table 1.2.

| Device Types | Sections | Support |
|---|---|---|
| BR/EDR | 1-8, 15-18 | C1 |
| LE-only | 1-3, 9-12, 15-18 | C1 |
| BR/EDR/LE | 1-18 | C1 |
| C1: Mandatory to support only one device type | | |

*Table 1.2: Requirements for device types*

## 1.2 SYMBOLS AND CONVENTIONS

### 1.2.1 Requirement Status Symbols

In this document (especially in the profile requirements tables), the following symbols are used:

‘M’ for mandatory to support (used for capabilities that shall be used in the profile);

’O’ for optional to support (used for capabilities that can be used in the profile);

‘C’ for conditional support (used for capabilities that shall be used in case a certain other capability is supported);

‘E’ for excluded within profile role (used for capabilities that may be supported by the unit but shall never be used in the profile role);

’N/A’ for not applicable (in the given context it is impossible to use this capability).

Some excluded capabilities are capabilities that, according to the relevant Bluetooth specification, are mandatory. These are features that may degrade operation of devices following this profile. Therefore, these features shall never be activated while a unit is operating as a unit within this profile.

In this specification, the word *shall* is used for mandatory requirements, the word *should* is used to express recommendations and the word *may* is used for options.

### 1.2.2 Signaling diagram conventions

The following arrows are used in diagrams describing procedures:

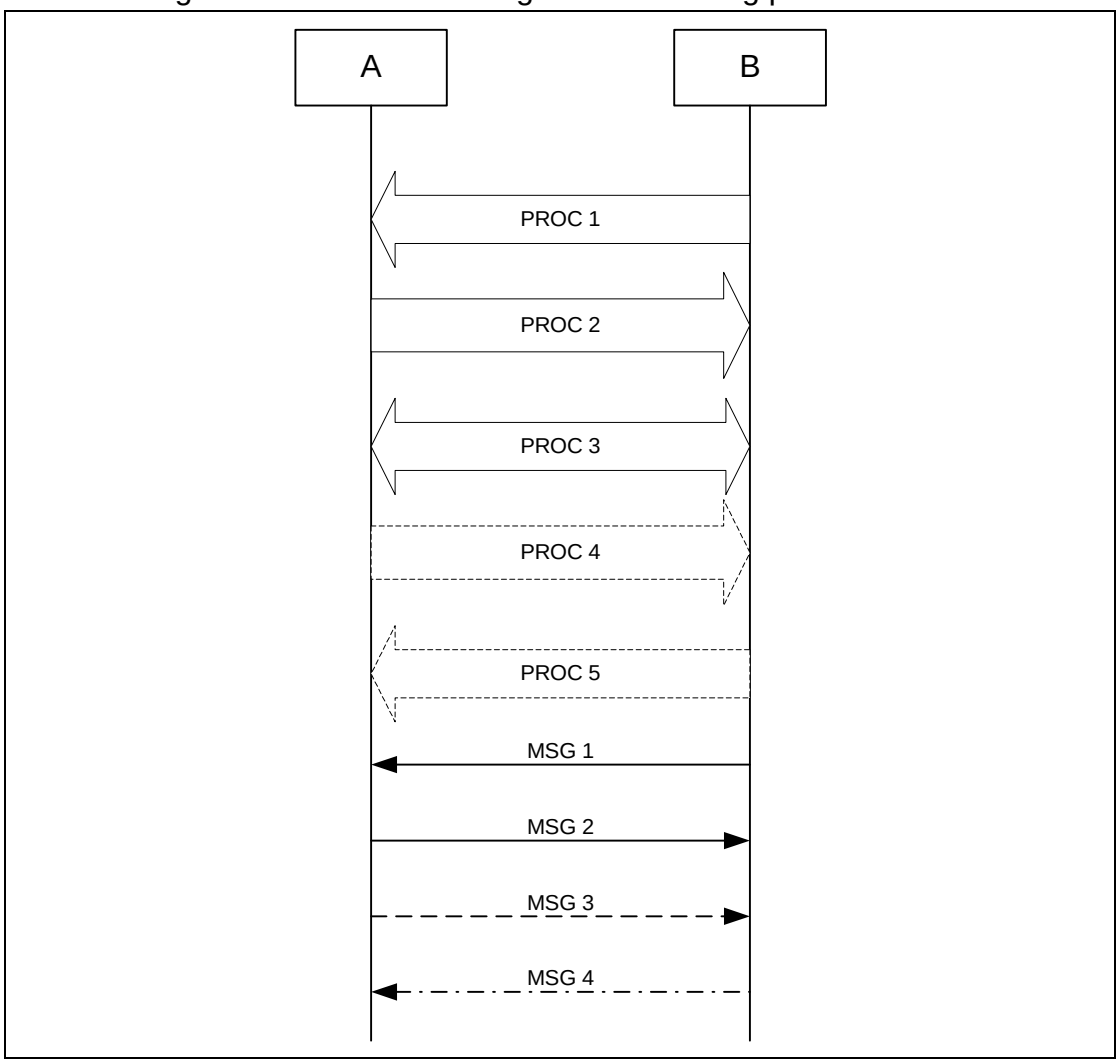



*Figure 1.1: Arrows used in signaling diagrams*

In Figure 1.1, the following cases are shown: PROC1 is a sub-procedure initiated by B. PROC2 is a sub-procedure initiated by A. PROC3 is a sub-procedure where the initiating side is undefined (may be both A or B). Dashed arrows denote optional steps. PROC4 indicates an optional sub-procedure initiated by A, and PROC5 indicates an optional sub-procedure initiated by B.

MSG1 is a message sent from B to A. MSG2 is a message sent from A to B. MSG3 indicates an optional message from A to B, and MSG4 indicates a conditional message from B to A.

### 1.2.3 Notation for Timers and Counters

Timers are introduced specific to this profile. To distinguish them from timers used in the Bluetooth protocol specifications and other profiles, these timers are named in the following format: '$T_{GAP}(nnn)$'.

# 2 PROFILE OVERVIEW

## 2.1 PROFILE STACK

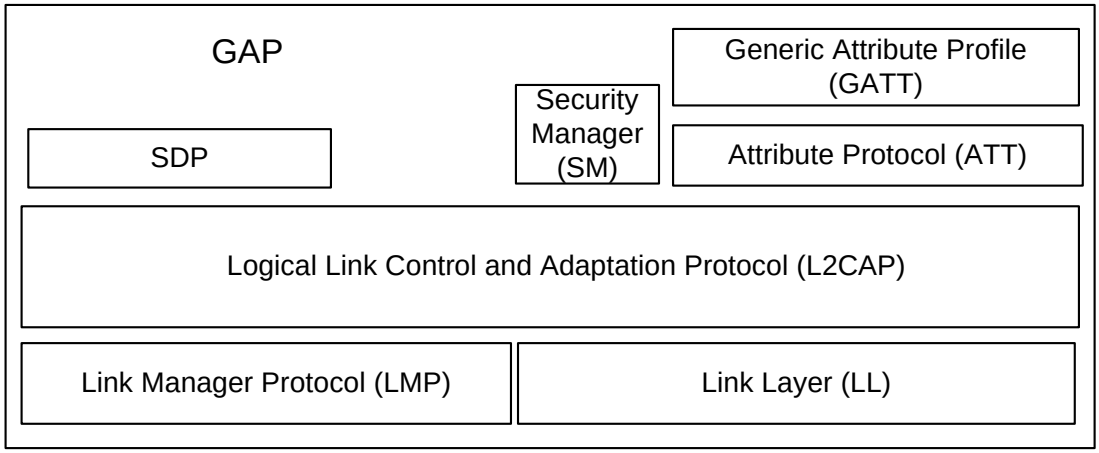



*Figure 2.1: Relationship of GAP with lower layers of the Bluetooth architecture*

The purpose of this profile is to describe:

- Profile roles
- Discoverability modes and procedures
- Connection modes and procedures
- Security modes and procedures

## 2.2 PROFILE ROLES

### 2.2.1 Roles when Operating over BR/EDR Physical Channel

For the descriptions in this profile of the roles that the two devices involved in a Bluetooth communication can take, the generic notation of the A-party (the *paging device* in case of link establishment, or *initiator* in case of another procedure on an established link) and the B-party (*paged device* or *acceptor*) is used. The A-party is the one that, for a given procedure, initiates the establishment of the physical link or initiates a transaction on an existing link.

This profile handles the procedures between two devices related to discovery and connecting (link and connection establishment) for the case where none of the two devices has any link established as well as the case where (at least) one device has a link established (possibly to a third device) before starting the described procedure.

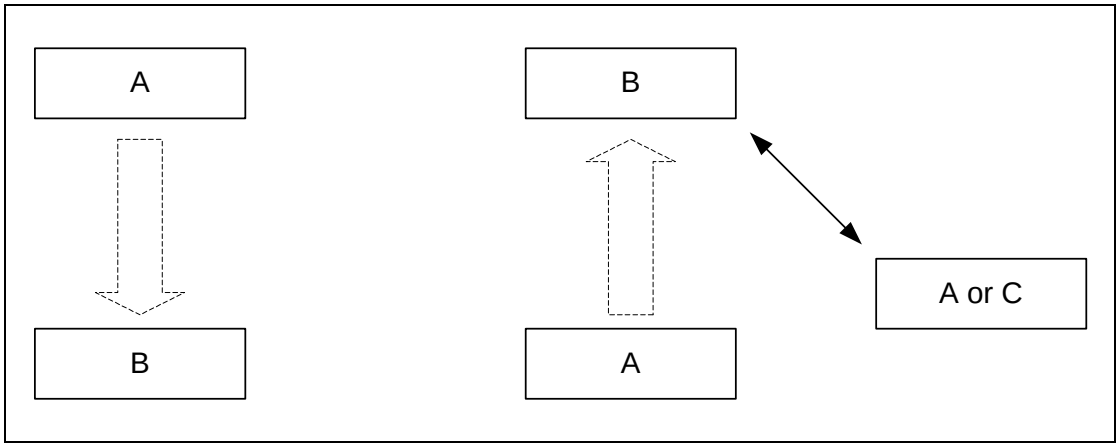



*Figure 2.2: This profile covers procedures initiated by one device (A) towards another device (B) that may or may not have an existing Bluetooth link active.*

The initiator and the acceptor generally operate the generic procedures according to this profile or another profile referring to this profile. If the acceptor operates according to several profiles simultaneously, this profile describes generic mechanisms for how to handle this.

### 2.2.2 Roles when Operating over an LE Physical Channel

There are four GAP roles defined for devices operating over an LE physical channel:

- Broadcaster
- Observer
- Peripheral
- Central

#### *2.2.2.1 Broadcaster Role*

A device operating in the Broadcaster role is a device that sends advertising events as described in [Vol. 6], Part B Section 4.4.2. A device operating in the Broadcaster role is referred to as a Broadcaster and shall have a transmitter and may have a receiver.

#### *2.2.2.2 Observer Role*

A device operating in the Observer role is a device that receives advertising events as described in [Vol. 6], Part B Section 4.4.3. A device operating in the Observer role is referred to as an Observer and shall have a receiver and may have a transmitter.

### 2.2.2.3 Peripheral Role

Any device that accepts the establishment of an LE physical link using any of the connection establishment procedures as defined in Section 13.3 is referred to as being in the Peripheral role. A device operating in the Peripheral role will be in the Slave role in the Link Layer Connection State as described in [Vol. 6], Part B Section 4.5. A device operating in the Peripheral role is referred to as a Peripheral. A Peripheral shall have both a transmitter and a receiver.

### 2.2.2.4 Central Role

A device that supports the Central role initiates the establishment of a physical connection. A device operating in the Central role will be in the Master role in the Link Layer Connection State as described in [Vol. 6], Part B Section 4.5. A device operating in the Central role is referred to as a Central. A Central shall have both a transmitter and a receiver.

### 2.2.2.5 Concurrent Operation in Multiple GAP Roles

A device may operate in multiple GAP roles concurrently if supported by the Controller. The Host should read the supported Link Layer States and State combinations from the Controller before any procedures or modes are used.

Only supported and allowed Link Layer States and State combinations may be used as described in [Vol. 6], Part B Section 1.1.1. The Physical Layer and Link Layer functionalities of each GAP role are shown in Table 2.1.

| | **GAP Roles when Operating over an LE Physical Channel** | | | |
|---|---|---|---|---|
| | **Broadcaster** | **Observer** | **Peripheral** | **Central** |
| **Physical Layer functionality** | | | | |
| • Transmitter Characteristics | M | O | M | M |
| • Receiver Characteristics | O | M | M | M |
| **Link Layer functionality** | | | | |
| States: | | | | |
| • Standby State | M | M | M | M |
| • Advertising State | M | E | M | E |
| • Scanning State | E | M | E | M |
| • Initiating State | E | E | E | M |
| Connection State: | | | | |

| | | | | |
|---|---|---|---|---|
| • Slave Role | E | E | M | E |
| • Master Role | E | E | E | M |
| Advertising event types: | | | | |
| • Connectable undirected event | E | E | M | E |
| • Connectable directed event | E | E | O | E |
| • Non-connectable undirected event | M | E | O | E |
| • Scannable undirected event | O | E | O | E |
| Scanning type: | | | | |
| • Passive scanning | E | M | E | O |
| • Active scanning | E | O | E | C1 |
| Link Layer control procedures: | | | | |
| • Connection Update procedure | E | E | M | M |
| • Channel Map Update procedure | E | E | M | M |
| • Encryption procedure | E | E | O | O |
| • Feature Exchange procedure | E | E | M | M |
| • Version Exchange procedure | E | E | M | M |
| • Termination procedure | E | E | M | M |
| C1: If passive scanning is supported then active scanning is optional, otherwise active scanning is mandatory. | | | | |

*Table 2.1: Physical Layer and Link Layer functionality for GAP roles when operating over an LE physical channel*

## 2.3 USER REQUIREMENTS AND SCENARIOS

The Bluetooth user should, where expected, be able to connect a Bluetooth device to any other Bluetooth device. Even if the two connected devices don't share any common application, it should be possible for the user to find this out using basic Bluetooth capabilities. When the two devices do share the same application but are from different manufacturers, the ability to connect them should not be blocked just because manufacturers choose to call basic Bluetooth capabilities by different names on the user interface level or implement basic procedures to be executed in different orders.

## 2.4 PROFILE FUNDAMENTALS

This profile states the requirements on names, values and coding schemes used for names of parameters and procedures experienced on the user interface level.

This profile defines modes of operation that are not service- or profile-specific, but that are generic and can be used by profiles referring to this profile, and by devices implementing multiple profiles.

This profile defines the general procedures that can be used for discovering identities, names and basic capabilities of other Bluetooth devices that are in a mode where they can be discovered. Only procedures where no channel or connection establishment is used are specified.

This profile defines the general procedure for how to create bonds (i.e., dedicated exchange of link keys) between Bluetooth devices.

This profile describes the general procedures that can be used for establishing connections to other Bluetooth devices that are in a mode that allows them to accept connections and service requests.

## 2.5 CONFORMANCE

Bluetooth devices shall conform to this profile to ensure basic interoperability.

Bluetooth devices that conform to another Bluetooth profile may use adaptations of the generic procedures as specified by that other profile. They shall, however, be compatible with devices compliant to this profile at least on the level of the supported generic procedures.

All capabilities indicated as mandatory for this profile shall be supported in the specified manner (process-mandatory). This also applies for all optional and conditional capabilities for which support is indicated. All mandatory capabilities, and optional and conditional capabilities for which support is indicated, are subject to verification as part of the Bluetooth certification program.

## 2.6 OTHER REQUIREMENTS

Devices of the BR/EDR or BR/EDR/LE device types shall implement and use SDP for service discovery over the BR/EDR transport. Devices of the BR/EDR devices type that implement the Attribute Protocol shall also implement the Generic Attribute Profile.

Devices of the BR/EDR/LE and LE-only device types shall implement and use GATT for service discovery of the LE transport. Devices of the LE-only device type shall implementation ATT.

# 3 USER INTERFACE ASPECTS

## 3.1 THE USER INTERFACE LEVEL

In the context of this specification, the user interface level refers to places (such as displays, dialog boxes, manuals, packaging, advertising, etc.) where users of Bluetooth devices encounters names, values and numerical representation of Bluetooth terminology and parameters.

This profile specifies the generic terms that should be used on the user interface level.

A device that supports discovery and connections over both the BR/EDR and LE physical channels should have a unified user interface that hides the differences between the underlying technologies from the user.

## 3.2 REPRESENTATION OF BLUETOOTH PARAMETERS

### 3.2.1 Bluetooth Device Address (BD_ADDR)

*3.2.1.1 Definition*

A device shall be identified by a Bluetooth device address. When the device is operating over a BR/EDR physical channel the device address shall be the BD_ADDR (see [Vol. 2], Part B Section 1.2). The device address is obtained from a remote device during the device discovery procedure (see Section 6.4 and Section 13.2.3).

The BD_ADDR shall be created in accordance with Section 9.2 (“48-bit universal LAN MAC addresses”) of the IEEE 802-2001 standard (http://standards.ieee.org/getieee802/download/802-2001.pdf) and using a valid Organizationally Unique Identifier (OUI) obtained from the IEEE Registration Authority (see http://standards.ieee.org/regauth/oui/forms/ and sections 9 and 9.1 of the IEEE 802-2001 specification).

*3.2.1.1.1 Bluetooth Device Address in an LE-only Device Type*

An LE-only device type shall use either a public or a random device address. The public device address shall be set to the BD_ADDR. A random device address is defined in Section 10.8. An LE-only device type may support the privacy feature as defined in Section 10.7 which requires the support of private device addresses as defined in Section 10.8.2.

#### *3.2.1.1.2 Bluetooth Device Address in a BR/EDR/LE Device Type*

A BR/EDR/LE device type shall use the BD_ADDR on the BR/EDR physical channel and a public device address over the LE physical channel. The public device address shall be set to the BD_ADDR. A BR/EDR/LE device type shall be capable of connecting to an LE-only or another BR/EDR/LE device type that is identified by a random device address.

#### *3.2.1.2 Term on user interface level*

When the Bluetooth address is referred to on UI level, the term Bluetooth Device Address' should be used.

#### *3.2.1.3 Representation*

On BB level the BD_ADDR is represented as 48 bits (see [Vol. 2], Part B Section 1.2). On the Link Layer the public and random device address are represented as 48-bit addresses.

On the UI level the Bluetooth address shall be represented as 12 hexadecimal characters, possibly divided into sub-parts separated by':'.
(E.g., '000C3E3A4B69' or '00:0C:3E:3A:4B:69'.) At UI level, any number shall have the MSB -> LSB (from left to right) 'natural' ordering (e.g., the number '16' shall be shown as '0x10').

### 3.2.2 Bluetooth Device Name (the user-friendly name)

#### *3.2.2.1 Definition*

The Bluetooth device name is the user-friendly name that a Bluetooth device-exposes to remote devices. For a device supporting the BR/EDR device type, the name is a character string returned in the LMP_name_res in response to an LMP_name_req. For a device supporting the LE-only device type, the name is a character string held in the Device Name characteristic as defined in Section 12.1.

#### *3.2.2.1.1 Bluetooth Device Name in a Device with BR/EDR/LE Device Type*

A BR/EDR/LE device type shall have a single Bluetooth device name which shall be identical irrespective of the physical channel used to perform the name discovery procedure.

For the BR/EDR physical channel the name is received in the LMP_name_res. For the LE physical channel the name can be read from the Device Name characteristic as defined in Section 12.1.

Note: The Device Name Characteristic of the local device can be read by a remote device using ATT over BR/EDR if the local device supports ATT over BR/EDR.

#### *3.2.2.2 Term on user interface level*

When the Bluetooth device name is referred to on UI level, the term ‘Bluetooth Device Name’ should be used.

#### *3.2.2.3 Representation*

The Bluetooth device name can be up to 248 bytes (see [Vol. 2], Part C Section 4.3.5). It shall be encoded according to UTF-8 (i.e., name entered on the UI level may be down to 82 characters outside the Unicode range 0x00-0x7F are used).

A device cannot expect that a general remote device is able to handle more than the first 40 characters of the Bluetooth device name. If a remote device has limited display capabilities, it may use only the first 20 characters.

### 3.2.3 Bluetooth Passkey (Bluetooth PIN)

#### *3.2.3.1 Definition*

The Bluetooth passkey may be used to authenticate two Bluetooth devices with each other during the creation of a mutual link key via the pairing procedure. The passkey may be used in the pairing procedures to generate the initial link key.

The PIN may be entered on the UI level but may also be stored in the device; e.g., in the case of a device without sufficient MMI for entering and displaying digits.

#### *3.2.3.2 Terms at user interface level*

When the Bluetooth PIN is referred to on UI level, the term ‘Bluetooth Passkey’ should be used.

#### *3.2.3.3 Representation*

There are a number of different representations of the Bluetooth passkey. At a high level there are two distinct representations: one used with the Secure Simple Pairing and Security Manager, and another used with legacy pairing (where it is generally referred to as the Bluetooth PIN).

For Secure Simple Pairing and Security Manager, the Bluetooth passkey is a 6-digit numerical value. It is represented as integer value in the range 0x00000000 – 0x000F423F (000000 to 999999). The numeric value may be used as the input to the Authentication Stage 1 for Secure Simple Pairing Passkey Entry (see [Vol. 2], Part H Section 7.2.3), or as the TK value in the Security Manager for the process defined in [Vol. 3], Part H Section 2.3.5.

For legacy pairing (see Section 17.2 on page 399), the Bluetooth PIN has different representations on different levels. PINBB is used on baseband level, and PINUI is used on user interface level. PINBB is the PIN used by [1] for calculating the initialization key during the Pairing Procedure.

PINUI is the character representation of the PIN that is entered on the UI level. The transformation from PINUI to PINBB shall be according to UTF-8. PINBB may be 128 bits (16 bytes).

PIN codes may be up to 16 characters. In order to take advantage of the full level of security all PINs should be 16 characters long. Variable PINs should be composed of alphanumeric characters chosen from within the Unicode range 0x00-0x7F. If the PIN contains any decimal digits these shall be encoded using the Unicode Basic Latin characters (e.g., code points 0x30 to 0x39) (Note 1).

For compatibility with devices with numeric keypads, fixed PINs shall be composed of only decimal digits, and variable PINs should be composed of only decimal digits.

If a device supports entry of characters outside the Unicode range 0x00-0x7F, other Unicode code points may be used (Note 2), except the Halfwidth and Fullwidth Forms from within the Unicode range FF00 - FFEF shall not be used (Note 3).

**Examples:**

| **User-entered code** | **Corresponding $PIN_{BB}$[0..length-1] (value as a sequence of octets in hexadecimal notation)** |
|---|---|
| ’0196554200906493’ | length = 16, value = 0x30 0x31 0x39 0x36 0x35 0x35 0x34 0x32 0x30 0x30 0x39 0x30 0x36 0x34 0x39 0x33 |
| ’Børnelitteratur’ | length = 16, value = 0x42 0xC3 0xB8 0x72 0x6e 0x65 0x6c 0x69 0x74 0x74 0x65 0x72 0x61 0x74 0x75 0x72 |

Note 1: This is to prevent interoperability problems since there are decimal digits at other code points (e.g., the Fullwidth digits at code points 0xff10 to 0xff19).

Note 2: Unicode characters outside the Basic Latin range (0x00 - 0x7F) encode to multiple bytes; therefore, when characters outside the Basic Latin range are used the maximum number of characters in the PINUI will be less than 16. The second example illustrates a case where a 16 character string encodes to 15 bytes because the character ø is outside the Basic Latin range and encodes to two bytes (0xC3 0xB8).

Note 3: This is to prevent interoperability problems since the Halfwidth and Fullwidth forms contain alternative variants of ASCII, Katakana, Hangul, punctuation and symbols. All of the characters in the Halfwidth and Fullwidth forms have other more commonly used Unicode code points.

### 3.2.4 Class of Device

#### *3.2.4.1 Definition*

Class of device is a parameter received during the device discovery procedure, indicating the type of device and which types of service that are supported.

#### *3.2.4.2 Term on user interface level*

The information within the Class of Device parameter should be referred to as ‘Bluetooth Device Class’ (i.e., the major and minor device class fields) and ‘Bluetooth Service Type’ (i.e., the service class field). The terms for the defined Bluetooth Device Types and Bluetooth Service Types are defined in [7].

When using a mix of information found in the Bluetooth Device Class and the Bluetooth Service Type, the term ‘Bluetooth Device Type’ should be used.

#### *3.2.4.3 Representation*

The Class of device is a bit field and is defined in [7]. The UI-level representation of the information in the Class of Device is implementation specific.

#### *3.2.4.4 Usage*

Some devices provide more than one service and a given service may be provided by different device types. Therefore, the device type does not have a one-to-one relationship with services supported. The major and minor device class field should not be used to determine whether a device supports any specific service(s). It may be used as an indication of devices that are most likely to support desired services before service discovery requests are made, and it may be used to guide the user when selecting among several devices that support the same service.

## 3.3 PAIRING

Pairing over a BR/EDR physical link is defined on LMP level (LMP pairing, see 17.2). Pairing over an LE physical link is defined by the Security Manager specification ([Vol. 3], Part H Section 2.3). Either the user initiates the bonding procedure and enters the passkey with the explicit purpose of creating a bond (and maybe also a secure relationship) between two Bluetooth devices, or the user is requested to enter the passkey during the establishment procedure since the devices did not share a common link key beforehand. In the first case, the user is said to perform “bonding (with entering of passkey)” and in the second case the user is said to “authenticate using the passkey.”

# 4 MODES

| Procedure | Ref. | Support |
|---|---|---|
| Discoverability modes: | 4.1 | |
| Non-discoverable mode | | C1 |
| Discoverable mode | | O |
| Limited discoverable mode | | C3 |
| General discoverable mode | | C4 |
| Connectability modes: | 4.2 | |
| Non-connectable mode | | O |
| Connectable mode | | M |
| Bondable modes: | 4.3 | |
| Non-bondable mode | | O |
| Bondable mode | | C2 |
| C1: If limited discoverable mode is supported, non-discoverable mode is mandatory, otherwise optional. | | |
| C2: If the bonding procedure is supported, support for bondable mode is mandatory, otherwise optional. | | |
| C3: If Discoverable mode is supported, support for Limited Discoverable mode is mandatory, otherwise optional. | | |
| C4: If Discoverable mode is supported, support for General Discoverable mode is mandatory, otherwise optional | | |

*Table 4.1: Conformance requirements related to modes defined in this section*

## 4.1 DISCOVERABILITY MODES

With respect to inquiry, a Bluetooth device shall be either in non-discoverable mode or in a discoverable mode. (The device shall be in one, and only one, discoverability mode at a time.) The two discoverable modes defined here are called limited discoverable mode and general discoverable mode. Inquiry is defined in [Vol. 2], Part B Section 8.4.

When a Bluetooth device is in non-discoverable mode it does not respond to inquiry.

A Bluetooth device is said to be made discoverable, or set into a discoverable mode, when it is in limited discoverable mode or in general discoverable mode. Even when a Bluetooth device is made discoverable, it may be unable to respond to inquiry due to other baseband activity (for example, reserved synchronous slots should have priority over response packets, so that synchro-

nous links may prevent a response from being returned). A Bluetooth device that does not respond to inquiry is called a silent device.

After being made discoverable, the Bluetooth device shall be discoverable for at least $T_{GAP}(103)$.

The speed of discovery is dependent on the configuration of the inquiry scan interval and inquiry scan type of the Bluetooth device. The Host is able to configure these parameters based on trade-offs between power consumption, bandwidth and the desired speed of discovery.

### 4.1.1 Non-discoverable Mode

#### *4.1.1.1 Definition*

When a Bluetooth device is in non-discoverable mode, it shall never enter the INQUIRY_SCAN state.

#### *4.1.1.2 Term on UI-level*

Bluetooth device is ‘non-discoverable’ or in ’non-discoverable mode’.

### 4.1.2 Limited Discoverable Mode

#### *4.1.2.1 Definition*

The limited discoverable mode should be used by devices that need to be discoverable only for a limited period of time, during temporary conditions, or for a specific event. The purpose is to respond to a device that makes a limited inquiry (inquiry using the LIAC).

A Bluetooth device should not be in limited discoverable mode for more than $T_{GAP}(104)$. The scanning for the limited inquiry access code can be done either in parallel or in sequence with the scanning of the general inquiry access code. When in limited discoverable mode, one of the following options shall be used.

- *Parallel scanning*

  When a Bluetooth device is in limited discoverable mode and when discovery speed is more important than power consumption or bandwidth, it is recommended that the Bluetooth device enter the INQUIRY_SCAN state at least every $T_{GAP}(105)$ and that Interlaced Inquiry scan is used.

  If, however, power consumption or bandwidth is important, but not critical, it is recommended that the Bluetooth device enter the INQUIRY_SCAN state at least every $T_{GAP}(102)$ and Interlaced Inquiry scan is used.

  When power consumption or bandwidth is critical it is recommended that the Bluetooth device enter the INQUIRY_SCAN state at least every $T_{GAP}(102)$ and Non Interlaced Inquiry scan is used.

In all cases the Bluetooth device shall enter the INQUIRY_SCAN state at least once in $T_{GAP}(102)$ and scan for the GIAC and the LIAC for at least $T_{GAP}(101)$.

When either a SCO or eSCO link is in operation, it is recommended to use interlaced scan to significantly decrease the discoverability time.

- *Sequential scanning*

  When a Bluetooth device is in limited discoverable mode, it shall enter the INQUIRY_SCAN state at least once in $T_{GAP}(102)$ and scan for the GIAC for at least $T_{GAP}(101)$ and enter the INQUIRY_SCAN state more often than once in $T_{GAP}(102)$ and scan for the LIAC for at least $T_{GAP}(101)$.

  If an inquiry message is received when in limited discoverable mode, the entry into the INQUIRY_RESPONSE state takes precedence over the next entries into INQUIRY_SCAN state until the inquiry response is completed.

#### *4.1.2.2 Conditions*

When a device is in limited discoverable mode it shall set bit no 13 in the Major Service Class part of the Class of Device/Service field [7].

#### *4.1.2.3 Term on UI-level*

Bluetooth device is ‘discoverable’ or in ‘discoverable mode’.

### 4.1.3 General Discoverable Mode

#### *4.1.3.1 Definition*

The general discoverable mode shall be used by devices that need to be discoverable continuously or for no specific condition. The purpose is to respond to a device that makes a general inquiry (inquiry using the GIAC).

#### *4.1.3.2 Conditions*

When a Bluetooth device is in general discoverable mode and when discovery speed is more important than power consumption or bandwidth, it is recommended that the Bluetooth device enter the INQUIRY_SCAN state at least every $T_{GAP}(105)$ and that Interlaced Inquiry scan is used.

If, however, power consumption or bandwidth is important, but not critical, it is recommended that the Bluetooth device enter the INQUIRY_SCAN state at least every $T_{GAP}(102)$ and Interlaced Inquiry scan is used.

When power consumption or bandwidth is critical it is recommended that the Bluetooth device enter the INQUIRY_SCAN state at least every $T_{GAP}(102)$ and Non-interlaced Inquiry scan is used.

In all cases the Bluetooth device shall enter the INQUIRY_SCAN state at least once in $T_{GAP}(102)$ and scan for the GIAC for at least $T_{GAP}(101)$.

When either a SCO or eSCO link is in operation, it is recommended to use interlaced scan to significantly decrease the discoverability time.

A device in general discoverable mode shall not respond to a LIAC inquiry.

#### *4.1.3.3 Term on UI-level*

Bluetooth device is ‘discoverable’ or in ‘discoverable mode’.

## 4.2 CONNECTABILITY MODES

With respect to paging, a Bluetooth device shall be either in non-connectable mode or connectable mode. Paging is defined in [Vol. 2], Part B Section 8.3.

When a Bluetooth device is in non-connectable mode it does not respond to paging. When a Bluetooth device is in connectable mode it responds to paging.

The speed of connections is dependent on the configuration of the page scan interval and page scan type of the Bluetooth device. The Host is able to configure these parameters based on trade-offs between power consumption, bandwidth and the desired speed of connection.

### 4.2.1 Non-connectable Mode

#### *4.2.1.1 Definition*

When a Bluetooth device is in non-connectable mode it shall never enter the PAGE_SCAN state.

#### *4.2.1.2 Term on UI-level*

Bluetooth device is ‘non-connectable’ or in ‘non-connectable mode’.

### 4.2.2 Connectable Mode

#### *4.2.2.1 Definition*

When a Bluetooth device is in connectable mode it shall periodically enter the PAGE_SCAN state. The device makes page scan using the Bluetooth device address, BD_ADDR. Connection speed is a trade-off between power consumption / available bandwidth and speed. The Bluetooth Host is able to make these trade-offs using the Page Scan interval, Page Scan window, and Interlaced Scan parameters.

R0 page scanning should be used when connection speeds are critically important and when the paging device has a very good estimate of the Bluetooth clock. Under these conditions it is possible for paging to complete within two times the page scan window. Because the page scan interval is equal to the page scan window it is not possible for any other traffic to go over the Bluetooth link when using R0 page scanning. In R0 page scanning it is not possible to use interlaced scan. R0 page scanning is the highest power consumption mode of operation.

When connection times are critical but the other device either does not have an estimate of the Bluetooth clock or when the estimate is possibly out of date, it is better to use R1 page scanning with a very short page scan interval, $T_{GAP}(106)$, and Interlaced scan. This configuration is also useful to achieve nearly the same connection speeds as R0 page scanning but using less power

consumption and leaving bandwidth available for other connections. Under these circumstances it is possible for paging to complete within $T_{GAP}(106)$. In this case the Bluetooth device shall page scan for at least $T_{GAP}(101)$.

When connection times are important but not critical enough to sacrifice significant bandwidth and/or power consumption it is recommended to use either $T_{GAP}(107)$ or $T_{GAP}(108)$ for the scanning interval. Using Interlaced scan will reduce the connection time by half but may use twice the power consumption. Under these circumstances it is possible for paging to complete within one or two times the page scanning interval depending on whether Interlaced Scan is used. In this case the Bluetooth device shall page scan for at least $T_{GAP}(101)$.

In all cases the Bluetooth device shall enter the PAGE_SCAN state at least once in $T_{GAP}(102)$ and scan for at least $T_{GAP}(101)$.

The page scan interval, page scan window size, and scan type for the six scenarios are described in Table 4.2:

| Scenario | Page Scan Interval | Page Scan Window | Scan Type |
|---|---|---|---|
| R0 (1.28s) | $T_{GAP}(107)$ | $T_{GAP}(107)$ | Normal scan |
| Fast R1 (100ms) | $T_{GAP}(106)$ | $T_{GAP}(101)$ | Interlaced scan |
| Medium R1 (1.28s) | $T_{GAP}(107)$ | $T_{GAP}(101)$ | Interlaced scan |
| Slow R1 (1.28s) | $T_{GAP}(107)$ | $T_{GAP}(101)$ | Normal scan |
| Fast R2 (2.56s) | $T_{GAP}(108)$ | $T_{GAP}(101)$ | Interlaced scan |
| Slow R2 (2.56s) | $T_{GAP}(108)$ | $T_{GAP}(101)$ | Normal scan |

*Table 4.2: Page scan parameters for connection speed scenarios*

When either a SCO or eSCO link is in operation, it is recommended to use interlaced scan to significantly decrease the connection time.

#### *4.2.2.2 Term on UI-level*

Bluetooth device is ‘connectable’ or in ‘connectable mode’.

## 4.3 BONDABLE MODES

With respect to bonding, a Bluetooth device shall be either in non-bondable mode or in bondable mode. In bondable mode the Bluetooth device accepts bonding initiated by the remote device, and in non-bondable mode it does not.

### 4.3.1 Non-bondable Mode

#### *4.3.1.1 Definition*

When a Bluetooth device is in non-bondable mode it shall not accept a pairing request that results in bonding. Devices in non-bondable mode may accept connections that do not request or require bonding.

A device in non-bondable mode shall respond to a received LMP_in_rand with LMP_not_accepted with the reason *pairing not allowed*. An HCI-compliant host stack shall respond to an HCI_PIN_Code_Request event with the HCI_PIN_Code_Request_Negative_Reply command.

When both devices support Secure Simple Pairing and are in non-bondable mode, the local host shall respond to an IO capability request with the Authentication_Requirements parameter requesting dedicated bonding or general bonding with a negative response. An HCI-compliant host stack shall respond to an HCI_IO_Capabilities_Request event with an HCI_IO_Capabilities_Request_Negative_Reply command.

#### *4.3.1.2 Term on UI-level*

Bluetooth device is ‘non-bondable’ or in ‘non-bondable mode’ or “does not accept bonding”.

### 4.3.2 Bondable Mode

#### *4.3.2.1 Definition*

When a Bluetooth device is in bondable mode, and Secure Simple Pairing is not supported by either the local or remote device, the local device shall respond to a received LMP_in_rand with LMP_accepted (or with LMP_in_rand if it has a fixed PIN). An HCI-compliant host stack shall respond to an HCI_PIN_Code_Request event with the HCI_PIN_Code_Request_Reply command.

When both devices support Secure Simple Pairing, the local host shall respond to a user confirmation request with a positive response. An HCI-compliant host stack shall respond to an HCI_User_Confirmation_Request event with an HCI_User_Confirmation_Request_Reply command or an HCI_User_Passkey_Request event with an HCI_User_Passkey_Request_Reply command.

#### *4.3.2.2 Term on UI-level*

Bluetooth device is ‘bondable’ or in ‘bondable mode’ or “accepts bonding”.

# 5 SECURITY ASPECTS

| | Procedure | Ref. | Support |
|---|---|---|---|
| 1 | Authentication | 5.1 | M |
| 2 | Security modes | 5.2 | |
| | Security mode 1 | | E |
| | Security mode 2 | | O.1 |
| | Security mode 3 | | E |
| | Security mode 4 | | M |
| O.1: Security Mode 2 may only be used for backwards compatibility when the remote device does not support Secure Simple Pairing. | | | |

*Table 5.1: Conformance requirements related to the generic authentication procedure and the security modes defined in this section*

## 5.1 AUTHENTICATION

### 5.1.1 Purpose

The generic authentication procedure describes how the LMP-authentication and LMP-pairing procedures are used when authentication is initiated by one Bluetooth device towards another, depending on if a link key exists or not and if pairing is allowed or not.

### 5.1.2 Term on UI level

‘Bluetooth authentication’.

### 5.1.3 Procedure

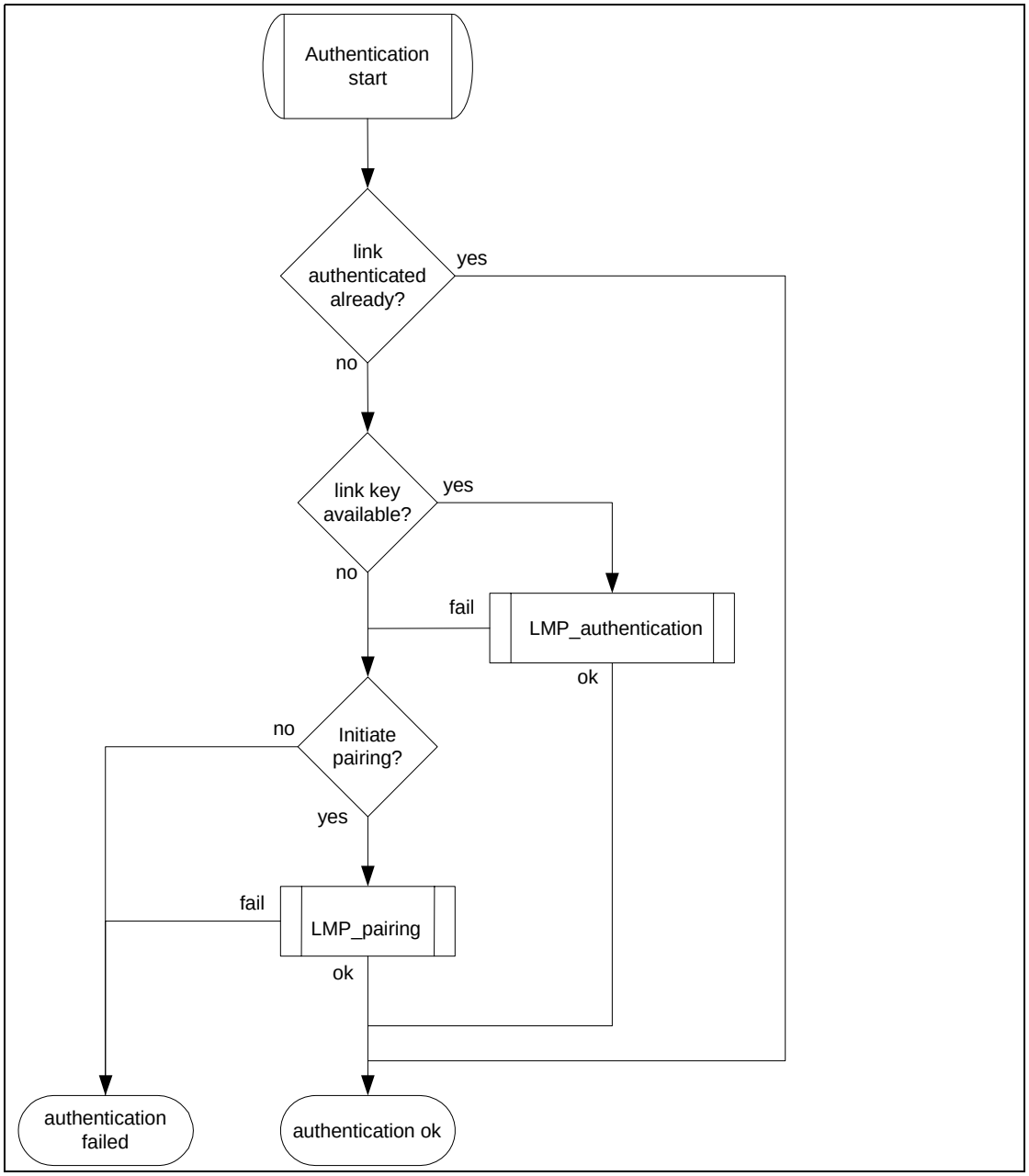



*Figure 5.1: Definition of the generic authentication procedure.*

### 5.1.4 Conditions

The local device shall initiate authentication after link establishment. The remote device may initiate security during or after link establishment.

## 5.2 SECURITY MODES

The following flow chart provides an overview of the channel establishment procedures.

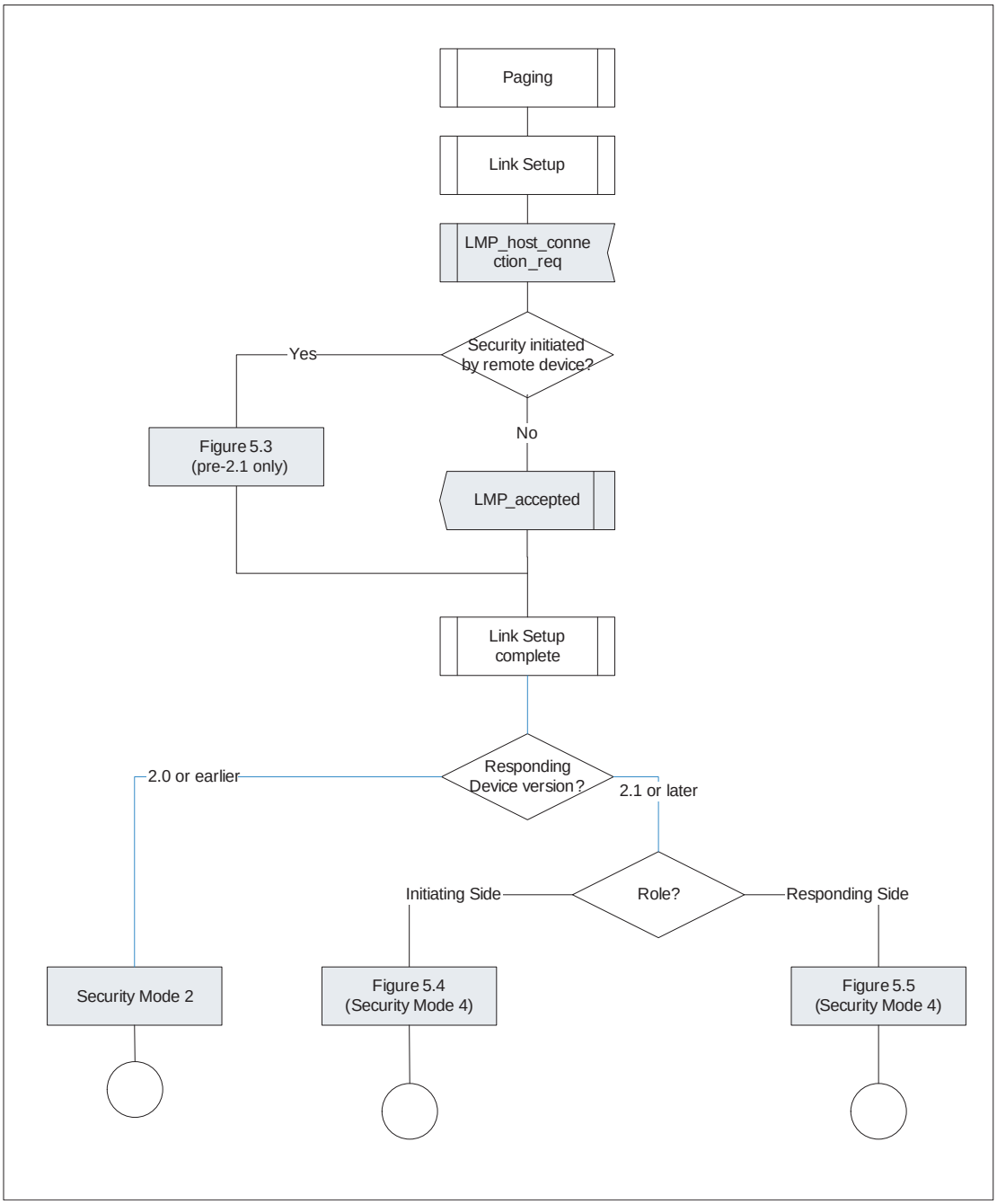



*Figure 5.2: Channel establishment with security.*

A device may support two security modes simultaneously: security mode 2 for backwards compatibility with remote devices that do not support Secure Simple Pairing and security mode 4 for devices that support Secure Simple Pairing.

### 5.2.1 Legacy Security Modes

Legacy security modes apply to those devices with a Controller or a Host that does not support SSP.

#### *5.2.1.1 Security mode 1 (non-secure)*

When a remote Bluetooth device is in security mode 1 it will never initiate any security procedure (i.e., it will never send LMP_au_rand, LMP_in_rand or LMP_encryption_mode_req).

#### *5.2.1.2 Security mode 2 (service level enforced security)*

When a remote Bluetooth device is in security mode 2 it will not initiate any security procedure before a channel establishment request (L2CAP_ConnectReq) has been received or a channel establishment procedure has been initiated by itself. (The behavior of a device in security mode 2 is further described in [6].) Whether a security procedure is initiated or not depends on the security requirements of the requested channel or service.

A Bluetooth device in security mode 2 should classify the security requirements of its services using at least the following attributes:

- Authorization required
- Authentication required
- Encryption required

*Note: Security mode 1 can be considered (at least from a remote device point of view) as a special case of security mode 2 where no service has registered any security requirements.*

#### *5.2.1.3 Security mode 3 (link level enforced security)*

When a remote Bluetooth device is in security mode 3 it will initiate security procedures before it sends LMP_setup_complete.

A Bluetooth device in security mode 3 may reject the host connection request (respond with LMP_not_accepted to the LMP_host_connection_req) based on settings in the host (e.g., only communication with pre-paired devices allowed).

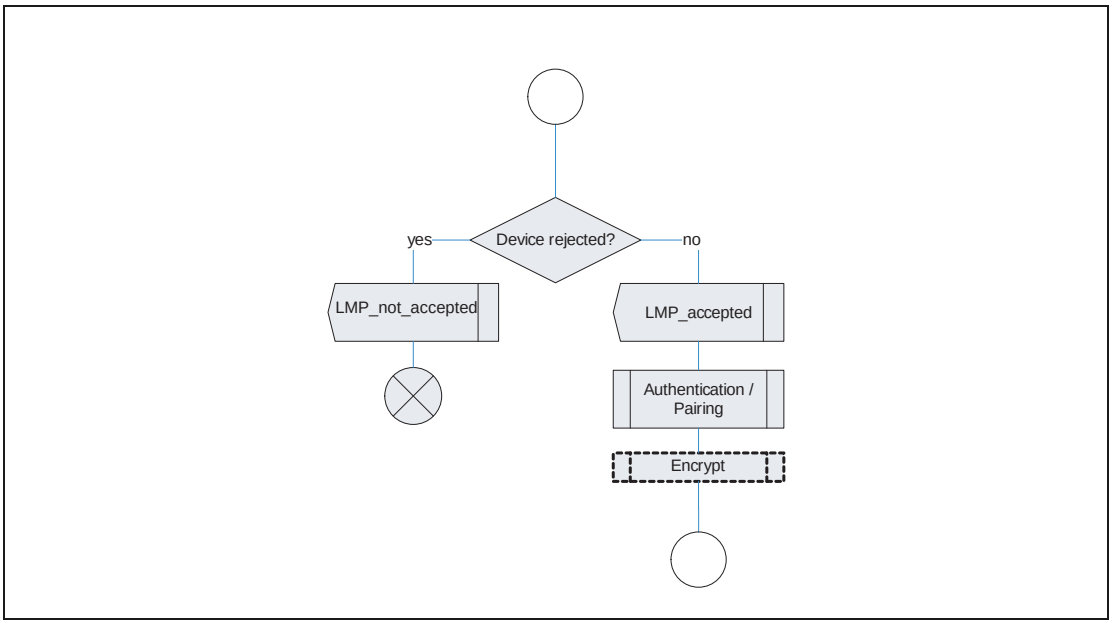



*Figure 5.3: Security mode 3 with a legacy remote device*

### 5.2.2 Security Mode 4 (service level enforced security)

A Bluetooth device in security mode 4 shall classify the security requirements of its services using at least the following attributes (in order of decreasing security):

- Authenticated link key required
- Unauthenticated link key required
- Security optional – SDP only. Limited to specific services (see Section 5.2.2.8).

An authenticated link key is a link key where either the numeric comparison, out-of-band, or passkey entry simple pairing association models were used. An authenticated link key has protection against man-in-the-middle (MITM) attacks. To ensure that an authenticated link key is created during the Simple Pairing procedure, the Authentication_Requirements parameter should be set to one of the MITM Protection Required options. An *unauthenticated link key* is a link key where the just works Secure Simple Pairing association model was used. An unauthenticated link key does not have protection against MITM attacks.

When both devices support Secure Simple Pairing, GAP shall require at least an unauthenticated link key and enabling encryption for all connections except those allowed to have security level 0 (see Section 5.2.2.8). A profile or protocol may define services that require more security (e.g., an authenticated link key) or no security (although unencrypted connections are only allowed when connecting to a service allowed to have security level 0). To allow an unauthenticated link key to be created during the Simple Pairing procedure, the

Authentication_Requirements parameter may be set to one of the MITM Protection Not Required options.

When the device is in Bondable Mode, it shall enable Secure Simple Pairing mode prior to entering Connectable Mode or establishing a link.

A Bluetooth device in security mode 4 shall respond to authentication requests during link establishment when the remote device is in security mode 3 for backwards compatibility reasons.

A Bluetooth device in security mode 4 enforces its security requirements before it attempts to access services offered by a remote device and before it grants access to services it offers to remote devices. Service access may occur via L2CAP channels or via channels established by protocols above L2CAP such as RFCOMM.

For services transmitting unicast data over the connectionless L2CAP channel, the transmitting device shall enforce its security requirements prior to sending data. There is no mechanism for a device receiving data via the L2CAP connectionless channel to prevent unencrypted data from being received. Hence, Section 5.2.2.1 addresses unicast connectionless data transmission together with devices initiating connection-oriented channels while Section 5.2.2.2 covers only devices responding to requests for connection-oriented channel establishment but does not cover unicast connectionless data reception.

#### *5.2.2.1 Security for Access to Remote Service (Initiating Side)*

When the responding device does not support Secure Simple Pairing, it may disconnect the link while the initiating device is requesting the PIN to be entered by the user. This may occur due to the lack of an L2CAP channel being present for longer than an implementation-specific amount of time (e.g., a few seconds). When this occurs, the initiating device shall allow the user to complete entering the PIN and shall then re-page the remote device and restart the pairing procedure (see [Vol. 2, Part C] Section 4.2.2 on page 249) using the PIN entered.

##### *5.2.2.1.1 Pairing Required for Access to Remote Service*

When a Bluetooth device in security mode 4 initiates access to a remote service via a connection-oriented L2CAP channel and a sufficient link-key is not available, the local device shall perform pairing procedures and enable encryption before sending a channel establishment request (L2CAP_ConnectReq or a higher-layer channel establishment request such as that of RFCOMM).

When a Bluetooth device in security mode 4 transmits data to a remote service via the unicast connectionless L2CAP channel and a sufficient link-key is not available, the local device shall perform pairing procedures and enable encryption before transmitting unicast data on the connectionless L2CAP channel.

See Section 5.2.2.8 on page 310 for details on determining whether or not a link key is sufficient.

If pairing does not succeed, the local device shall not send a channel establishment request. The local device may re-try pairing up to three (3) times. If pairing fails three consecutive times, the local device shall disconnect the ACL link with error code 0x05 - Authentication Failure.

If the link key generated is not at least as good as the expected or required type, the local device shall fail the establishment and may disconnect the ACL link with error code 0x05 - Authentication Failure.

#### *5.2.2.1.2 Authentication Required for Access to Remote Service*

When a Bluetooth device in security mode 4 initiates access to a remote service via a connection-oriented L2CAP channel and a sufficient link key is available for the remote device, it shall authenticate the remote device and enable encryption before sending a channel establishment request (L2CAP_ConnectReq or higher a layer-channel establishment request such as that of RFCOMM).

When a Bluetooth device in security mode 4 transmits unicast data to a remote service via the connectionless L2CAP channel and security is required for the application and a sufficient link-key is available then the local device shall authenticate the remote device and enable encryption before transmitting unicast data on the L2CAP connectionless channel.

See Section 5.2.2.8 on page 310 for details on determining whether or not a link key is sufficient.

If authentication is required by the service but does not succeed, or if a sufficient link-key is not available, then the local device shall not enable encryption and shall not send a channel establishment request and shall not send any unicast data via the L2CAP connectionless channel for that application. The host may then notify the user and offer to perform pairing.

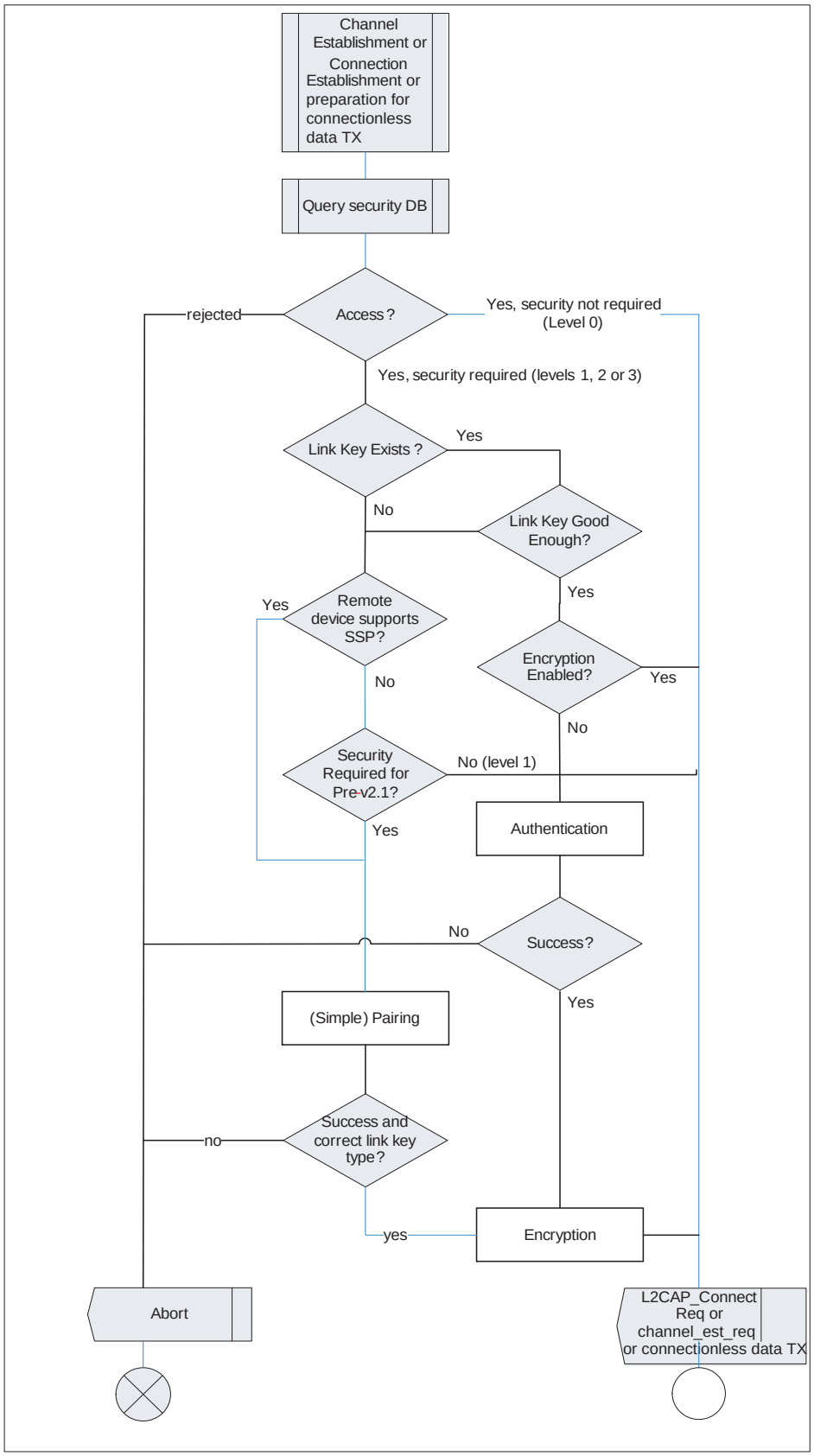



*Figure 5.4: Channel establishment using security mode 4 for initiating side*

#### *5.2.2.2 Security for Access to Local Service by Remote Device (Responding Side)*

When a remote device attempts to access a service offered by a Bluetooth device that is in security mode 4 and a sufficient link key exists and authentication has not been performed the local device shall authenticate the remote device and enable encryption after the channel establishment request is received but before a channel establishment confirmation (L2CAP_ConnectRsp with result code of 0x0000 or a higher-level channel establishment confirmation such as that of RFCOMM) is sent.

When L2CAP is the channel establishment protocol being used for the requested service, an L2CAP_ConnectRsp signaling packet shall be sent by the responding device containing the result code 0x0001 (connection pending) following receipt of an L2CAP_ConnectReq and prior to initiating security procedures which can result in prompting the local user for input (e.g., pairing using a PIN or Secure Simple Pairing using either the Passkey entry or Numerical Comparison association models). This will stop the L2CAP RTX timer on the remote device (which may be as short as 1 second) and will invoke the ERTX timer on the remote device, which is a minimum duration of 60 seconds.

See [Vol. 3, Part A] Section 6.1.7 on page 110, for additional information on L2CAP RTX and ERTX timers. See also [Vol. 3, Part A] Section 4.3 on page 59 for additional information on the L2CAP_ConnectRsp signalling packet, and the defined result codes.

Higher layer channel establishment protocols should be designed to restrict timeouts to be 30 seconds or longer to allow for user input, or provide mechanisms to dynamically extend timeouts when user input may be required.

If authentication or pairing fails when a remote device is attempting to access a local service, the local device shall send a negative response to the channel establishment request (L2CAP_ConnectReq or channel_est_req) indicating a security issue if possible. If the channel establishment protocol is L2CAP, then the result code 0x0003 (connection refused - security block) shall be sent in the L2CAP_ConnectRsp signal.

If the remote device has indicated support for Secure Simple Pairing, a channel establishment request is received for a service other than SDP, and encryption has not yet been enabled, then the local device shall disconnect the ACL link with error code 0x05 - Authentication Failure.

##### *5.2.2.2.1 Pairing Required for Access to Local Service by Remote Device*

When a remote device attempts to access a service offered by a Bluetooth device that is in security mode 4 and pairing is required due to the link key being absent or insufficient, the local device shall perform pairing procedures and enable encryption after the channel establishment request is received and before a channel establishment confirmation (L2CAP_ConnectRsp with result

code of 0x0000 or a higher-level channel establishment response such as that of RFCOMM) is sent.

See Section 5.2.2.6 for details on determining whether or not a link key is sufficient.

If pairing does not succeed, then the local device shall not send a channel establishment confirmation. The local device may retry pairing up to three (3) times. If pairing fails three consecutive times, then the local device shall disconnect the ACL link with error code 0x05 - Authentication Failure.

If the link-key generated is not at least as good as the expected or required type, then the local device shall fail the channel establishment and may disconnect the ACL link with error code 0x05 - Authentication Failure.

#### *5.2.2.2.2 Authentication Required for Access to Local Service by Remote Device*

See Section 5.2.2.6 for details on determining whether or not a link key is sufficient.

If authentication does not succeed, then the local device shall not send a channel establishment confirmation. The host may at this point notify the user and offer to perform pairing.

A Bluetooth device in security mode 4 shall respond to authentication and pairing requests during link establishment when the remote device is in security mode 3 for backwards compatibility reasons. However, authentication of the remote device shall be performed after the receipt of the channel establishment request is received, and before the channel establishment response is sent.

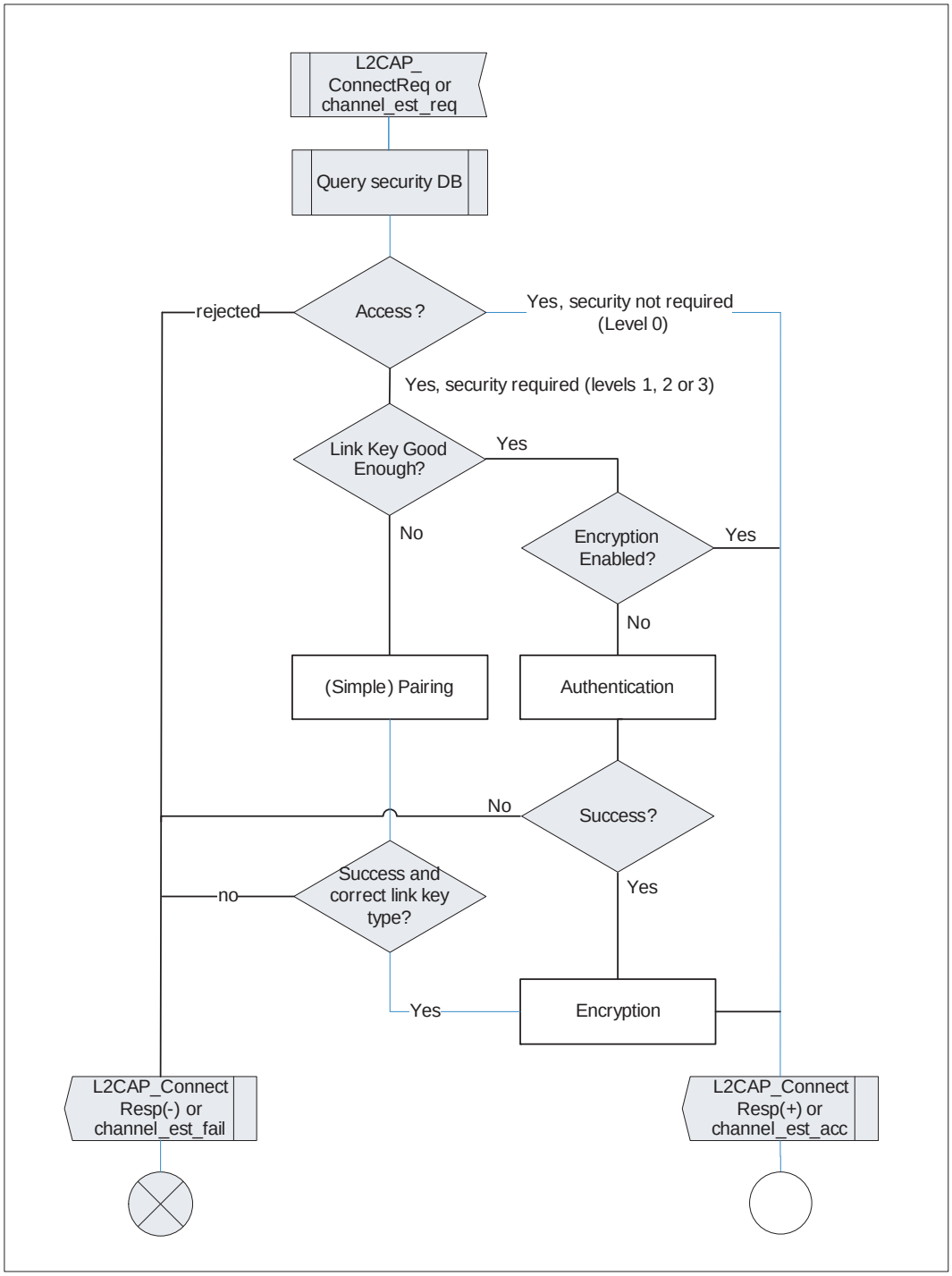



*Figure 5.5: Channel establishment using security mode 4 for the responding side*

### *5.2.2.3 Simple Pairing after Authentication Failure*

When both devices support Secure Simple Pairing all non-SDP connections are encrypted regardless of whether security was required or whether the devices are bonded or not. The initial connection between the two devices will result in a link key through Secure Simple Pairing. Depending on whether or not bonding was performed and the security policy of the initiating device, the

link key may or may not be stored. When the link key is stored, subsequent connections to the same device will use authentication but this may fail if the remote device has deleted the link key. Table 5.1 defines what shall be done depending on the type of the link key and whether bonding was performed or not.

| Link Key Type | Devices Bonded? | Action to take when Authentication Fails |
|---|---|---|
| Combination | No | Depends on security policy of the device:<br>• Option 1: Automatically initiate pairing<br>• Option 2: Notify user and ask if pairing is ok<br>Option 2 is recommended. |
| Combination | Yes | Notify user of security failure |
| Unauthenticated | No | Depends on security policy of the device:<br>• Option 1: Automatically initiate secure simple pairing<br>• Option 2: Notify user and ask if secure simple pairing is ok.<br>Option 1 is recommended. |
| Unauthenticated | Yes | Notify user of security failure |
| Authenticated | No | Depends on security policy of the device:<br>• Option 1: Automatically initiate secure simple pairing<br>• Option 2: Notify user and ask if secure simple pairing is ok<br>Option 2 is recommended. |
| Authenticated | Yes | Notify user of security failure |

*Table 5.1: Simple Pairing after Authentication Failure*

Note that non-bonded authenticated or unauthenticated link keys may be considered disposable by either device and may be deleted at any time.

### *5.2.2.4 IO Capabilities*

Once a connection is established, if the Host determines that security is necessary and both devices support Secure Simple Pairing, the devices perform an IO capability exchange. The purpose of the IO capability exchange is to determine the authentication algorithm used in the Authentication Stage 1 phase of Simple Pairing.

The input and output capabilities are described in Table 5.2:

| Capability | Description |
|---|---|
| No input | Device does not have the ability to indicate 'yes' or 'no' |

*Table 5.2: User Input Capabilities*

| Capability | Description |
|---|---|
| Yes / No | Device has at least two buttons that are mapped easily to 'yes' and 'no' or the device has a mechanism whereby the user can indicate either 'yes' or 'no' (see note below). |
| Keyboard | Device has a numeric keyboard that can input the numbers '0' through '9' and a confirmation. Device also has two buttons that can be easily mapped to 'yes' and 'no' or the device has a mechanism whereby the user can indicate either 'yes' or 'no' (see **Note** below). |

*Table 5.2: User Input Capabilities*

**Note**: 'yes' could be indicated by pressing a button within a certain time limit otherwise 'no' would be assumed.

| Capability | Description |
|---|---|
| No output | Device does not have the ability to display or communicate a 6 digit decimal number |
| Numeric output | Device has the ability to display or communicate a 6 digit decimal number |

*Table 5.3: User Output Capabilities*

### *5.2.2.5 Mapping of Input / Output Capabilities to IO Capability*

The individual input and output capabilities are mapped to a single IO capability which is later used to determine which authentication algorithm will be used.

| Local Output Capacity / Local Input Capacity | No Output | Numeric Output |
|---|---|---|
| **No input** | NoInputNoOutput | DisplayOnly |
| **Yes / No** | NoInputNoOutput | DisplayYesNo |
| **Keyboard** | KeyboardOnly | DisplayYesNo |

*Table 5.4: IO Capability mapping*

When a device has OOB authentication information from the remote device, it will indicate it in the LMP_IO_Capability_Res PDU. When either device has OOB information, the OOB authentication method will be used.

The Host may allow the Link Manager to ignore the IO capabilities and use the Numeric Comparison protocol with automatic accept by setting the Authentication_Requirements parameter to one of the MITM Protection *Not Required* options.

### 5.2.2.6 IO and OOB Capability Mapping to Authentication Stage 1 Method

Determining which association model to use in Authentication Stage 1 is performed in three steps. First, the devices look at the OOB Authentication Data Present parameter received in the remote IO capabilities. If either device has received OOB authentication data then the OOB association model is used. The event of receiving the OOB information is indicated by a device to its peer in the IO Capability Exchange step of simple pairing.

| Device A / Device B | **Has not received remote OOB authentication data** | **Has received remote OOB authentication table** |
|---|---|---|
| **Has not received remote OOB authentication data** | Use the IO capability mapping table | Use OOB association with<br>ra = 0<br>rb from OOB |
| **Has received remote OOB authentication data** | Use OOB association with<br>ra from OOB<br>rb = 0 | Use OOB association with<br>ra from OOB<br>rb from OOB |

*Table 5.5: IO and OOB capability mapping*

Second, if neither device has received OOB authentication data and if both devices have set the Authentication_Requirements parameter to one of the MITM Protection Not Required options, authentication stage 1 shall function as if both devices set their IO capabilities to DisplayOnly (e.g., Numeric comparison with automatic confirmation on both devices).

Finally, if neither device has received OOB authentication data and if one or both devices have set the Authentication_Requirements parameter to one of the *MITM Protection Required* options, the IO and OOB capabilities are mapped to the authentication stage 1 method as defined in the following table. A Host that has set the Authentication_Requirements parameter to one of the *MITM Protection Required* options shall verify that the resulting Link Key is an Authenticated Combination Key (see Section 7.7.24 in Host Controller Interface Functional Specification). The table also lists whether the combination key results in an authenticated or unauthenticated link key.

Note: The "DisplayOnly" IO capability only provides unidirectional authentication.

### 5.2.2.7 Out of Band (OOB)

An out of band mechanism may also be used to communicate discovery information as well as other information related to the pairing process.

The contents of the OOB data block are:

Mandatory contents:

| Initiator A / B Responder | Display Only | DisplayYesNo | KeyboardOnly | NoInputNoOutput |
|---|---|---|---|---|
| **DisplayOnly** | Numeric Comparison with automatic confirmation on both devices.<br><br>Unauthenticated | Numeric Comparison with automatic confirmation on device B only.<br><br>Unauthenticated | Passkey Entry: Responder Display, Initiator Input.<br><br>Authenticated | Numeric Comparison with automatic confirmation on both devices.<br><br>Unauthenticated |
| **DisplayYesNo** | Numeric Comparison with automatic confirmation on device A only.<br><br>Unauthenticated | Numeric Comparison: Both Display, Both Confirm.<br><br>Authenticated | Passkey Entry: Responder Display, Initiator Input.<br><br>Authenticated | Numeric Comparison with automatic confirmation on device A only.<br><br>Unauthenticated |
| **Keyboard Only** | Passkey Entry: Initiator Display, Responder Input.<br><br>Authenticated | Passkey Entry: Initiator Display, Responder Input.<br><br>Authenticated | Passkey Entry: Initiator and Responder Input.<br><br>Authenticated | Numeric Comparison with automatic confirmation on both devices.<br><br>Unauthenticated |
| **NoInputNoOutput** | Numeric Comparison with automatic confirmation on both devices.<br><br>Unauthenticated | Numeric Comparison with automatic confirmation on device B only.<br><br>Unauthenticated | Numeric Comparison with automatic confirmation on both devices.<br><br>Unauthenticated | Numeric Comparison with automatic confirmation on both devices.<br><br>Unauthenticated |

*Table 5.6: IO Capability Mapping to Authentication Stage 1*

- Length (2 bytes)
- BD_ADDR (6 bytes)

Optional contents:

- Class of Device (3 bytes)
- Simple Pairing Hash C (16 bytes)
- Simple Pairing Randomizer R (16 bytes)
- Local name (variable length)
- Other information

The length field includes all bytes in the OOB data block including the length field itself. The BD_ADDR will be a fixed field in the beginning of the OOB data block. Following the BD_ADDR will be zero or more EIR tag fields containing optional contents. The EIR tag format will be used for the optional contents.

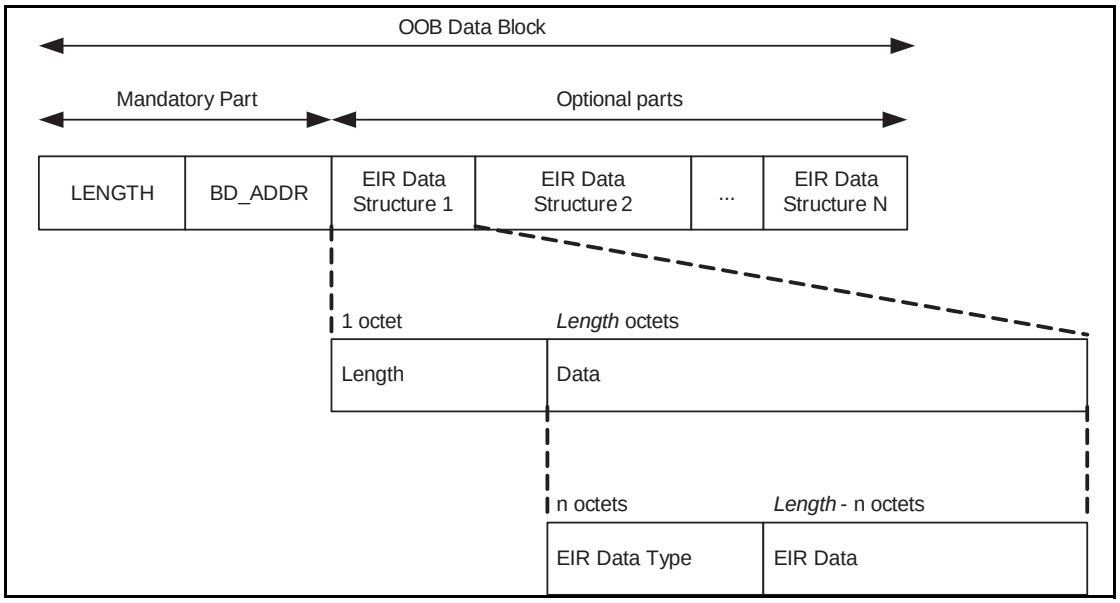



*Figure 5.6: OOB Data Block Format*

If Simple Pairing fails when one or both devices have OOB Authentication Data present, both devices shall discard the OOB Authentication Data and the device that originally initiated authentication shall re-initiate authentication. Note that although the user may get involved in authentication as defined by the IO capabilities of the two devices, falling back to the in-band authentication method will prevent deadlock conditions when one or both devices has stale OOB Authentication Data.

### *5.2.2.8 Security Database*

A Bluetooth device in security mode 4 shall classify the security requirements of its services using at least the following levels attributes (in order of decreasing security) for use when pairing with remote devices supporting Secure Simple Pairing:

- Level 3, for services with the following attributes:
  MITM protection required
  Encryption required
  User interaction acceptable
- Level 2, for services with the following attributes:
  MITM protection not required
  Encryption required
- Level 1, for services with the following attributes:

MITM protection not required

Minimal user interaction desired

One additional level is permitted for L2CAP fixed signaling channels (CIDs 0x0001, 0x0003 and 0x003F),SDP and for some service traffic sent via the connectionless L2CAP channel for services that do not require security but shall not be used for any other service traffic:

- Level 0: Service requires the following:

  MITM protection not required

  No encryption required

  No user interaction required

  Level 0 may only be used for Services with the following Service Class UUIDs:

  0x1000 + BASE_ID (Service Discovery Server)

The security level required for each service offered should be stored in a security database that is accessed to determine the type of link key that is required for access to the respective service. The security level required for service data transmitted on an L2CAP connection-oriented channel may differ from the security level required for service data transmitted on another L2CAP connection-oriented channel or on the connectionless L2CAP channel. Table 5.7 on page 311 shows the type of link key required for each security level for both remote devices that support Secure Simple Pairing (v2.1 + EDR remote devices) and for those that do not (pre-v2.1 + EDR remote devices).

| **Security Level Required for Service** | **Link Key type required for remote devices** | **Link Key type required for pre-v2.1 remote devices** | **Comments** |
|---|---|---|---|
| Level 3<br>• Strong MITM protection desired<br>• Encryption desired<br>• User interaction acceptable | Authenticated | Combination (16 digit PIN recommended) | High Security |
| Level 2<br>• MITM protection not necessary<br>• Encryption desired | Unauthenticated | Combination | Medium Security |

*Table 5.7: Security Level mapping to link key requirements*

| Security Level Required for Service | Link Key type required for remote devices | Link Key type required for pre-v2.1 remote devices | Comments |
|---|---|---|---|
| Level 1<br>• MITM protection not necessary<br>• Encryption not necessary[1]<br>• Minimal user interaction desired | Unauthenticated | None | Low Security |
| Level 0<br>• MITM protection not necessary<br>• Encryption not necessary<br>• No user interaction desired | None | None | Permitted only for SDP and service data sent via either L2CAP fixed signaling channels or the L2CAP connectionless channel to PSMs that correspond to service class UUIDs which are allowed to utilize Level 0. |

*Table 5.7: Security Level mapping to link key requirements*

1. Though encryption is not necessary for the service for Level 1, this specification mandates the use of encryption when the remote device is v2.1+EDR for all services other than SDP.

An *authenticated* link key is a link key where either the numeric comparison, out-of-band, or passkey entry simple pairing association models were used. An authenticated link key has protection against MITM attacks. To ensure that an authenticated link key is created during the Simple Pairing procedure, the Authentication_Requirements parameter should be set to one of the *MITM Protection Required* options.

An *unauthenticated* link key is a link key where the “Just Works” simple pairing association model was used (see [Vol. 1, Part A] Section 5.1.4 on page 87). An unauthenticated link key does not have protection against MITM attacks. To allow an unauthenticated link key to be created during the Simple Pairing procedure, the Authentication_Requirements parameter may be set to one of the *MITM Protection Not Required* options.

A combination link key is a link key where the v2.0 pairing mechanism was used to generate the link-key (see [Vol. 2, Part C] Section 4.2.2.4 on page 251).

When both devices support Secure Simple Pairing, GAP shall require at least an unauthenticated link key and enable encryption for service traffic sent or received via connection-oriented L2CAP channels. A profile or protocol may define services that require more security (for example, an authenticated link

key) or no security in the case of SDP or service traffic sent via the L2CAP connectionless channel for services that do not require security.

When the device is in Bondable Mode, it shall enable Secure Simple Pairing mode prior to entering Connectable Mode or establishing a link.

A Bluetooth device in security mode 4 shall respond to authentication and pairing requests during link establishment when the remote device is in security mode 3 for backwards compatibility reasons. See Section 5.2.1.3 for more information.

The remote Controller's and remote Host's support for Secure Simple Pairing shall be determined by the Link Manager Secure Simple Pairing (Host Support) feature bit.

A previously generated link key is considered “sufficient” if the link key type is of the type required for the service, or of a higher strength. Authenticated link keys are considered higher strength than Unauthenticated or Combination keys. Unauthenticated link keys are considered higher strength than Combination keys.

# 6 IDLE MODE PROCEDURES

The inquiry and discovery procedures described here are applicable only to the device that initiates them (A). The requirements on the behavior of B is according to the modes specified in 4 and to [2].

| | Procedure | Ref. | Support |
|---|---|---|---|
| 1 | General inquiry | 6.1 | C1 |
| 2 | Limited inquiry | 6.2 | C1 |
| 3 | Name discovery | 6.3 | O |
| 4 | Device discovery | 6.4 | O |
| 5 | Bonding | 6.5 | O |

C1: If initiation of bonding is supported, support for simple inquiry procedure is mandatory, otherwise optional.

(Note: support for LMP-pairing is mandatory [2].)

## 6.1 GENERAL INQUIRY

### 6.1.1 Purpose

The purpose of the general inquiry procedure is to provide the initiator with the Bluetooth device address, clock, Class of Device, page scan mode, and extended inquiry response information of general discoverable devices (i.e., devices that are in range with regard to the initiator and are set to scan for inquiry messages with the General Inquiry Access Code). Also devices in limited discoverable mode will be discovered using general inquiry.

The general inquiry should be used by devices that need to discover devices that are made discoverable continuously or for no specific condition.

### 6.1.2 Term on UI level

‘Bluetooth Device Inquiry’.

### 6.1.3 Description

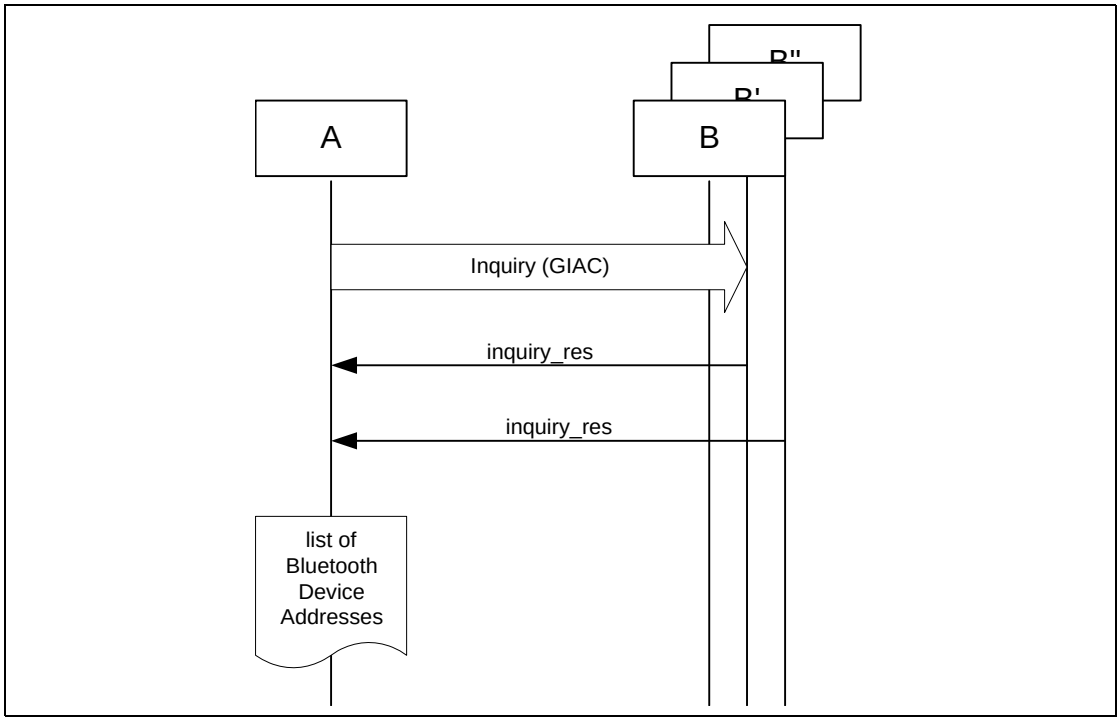



*Figure 6.1: General inquiry, where B is a device in non-discoverable mode, B´ is a device in limited discoverable mode and B" is a device in general discoverable mode. (Note that all discoverable devices are discovered using general inquiry, independent of which discoverable mode they are in.)*

### 6.1.4 Conditions

When general inquiry is initiated by a Bluetooth device, the INQUIRY state shall last $T_{GAP}(100)$ or longer, unless the inquirer collects enough responses and determines to abort the INQUIRY state earlier. The Bluetooth device shall perform inquiry using the GIAC.

In order for Device A to receive inquiry responses, the remote devices in range have to be made discoverable (limited or general).

## 6.2 LIMITED INQUIRY

### 6.2.1 Purpose

The purpose of the limited inquiry procedure is to provide the initiator with the Bluetooth device address, clock, Class of Device, page scan mode, and extended inquiry response information of limited discoverable devices. The latter devices are devices that are in range with regard to the initiator, and may be set to scan for inquiry messages with the Limited Inquiry Access Code, in addition to scanning for inquiry messages with the General Inquiry Access Code.

The limited inquiry should be used by devices that need to discover devices that are made discoverable only for a limited period of time, during temporary conditions or for a specific event. Since it is not guaranteed that the discoverable device scans for the LIAC, the initiating device may choose any inquiry procedure (general or limited). Even if the remote device that is to be discovered is expected to be made limited discoverable (e.g., when a dedicated bonding is to be performed), the limited inquiry should be done in sequence with a general inquiry in such a way that both inquiries are completed within the time the remote device is limited discoverable; i.e., at least $T_{GAP}(103)$.

### 6.2.2 Term on UI level

'Bluetooth Device Inquiry'.

### 6.2.3 Description

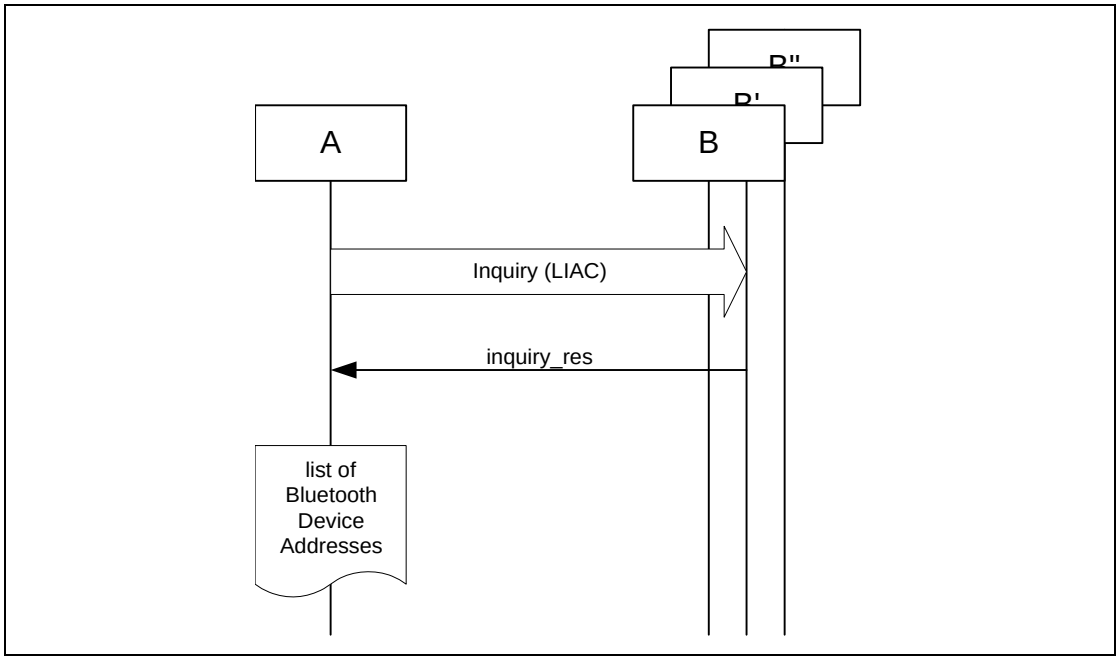



*Figure 6.2: Limited inquiry where B is a device in non-discoverable mode, B' is a device in limited discoverable mode and B" is a device in general discoverable mode. (Note that only limited discoverable devices can be discovered using limited inquiry.)*

### 6.2.4 Conditions

When limited inquiry is initiated by a Bluetooth device, the INQUIRY state shall last $T_{GAP}(100)$ or longer, unless the inquirer collects enough responses and determines to abort the INQUIRY state earlier. The Bluetooth device shall perform inquiry using the LIAC.

In order for Device A to receive inquiry responses, the remote devices in range has to be made limited discoverable.

## 6.3 NAME DISCOVERY

### 6.3.1 Purpose

The purpose of name discovery is to provide the initiator with the Bluetooth Device Name of connectable devices (i.e., devices in range that will respond to paging).

### 6.3.2 Term on UI level

'Bluetooth Device Name Discovery'.

### 6.3.3 Description

#### *6.3.3.1 Name request*

Name request is the procedure for retrieving the Bluetooth Device Name from a connectable Bluetooth device. It is not necessary to perform the full link establishment procedure (see 7.1) in order to just to get the name of another device.

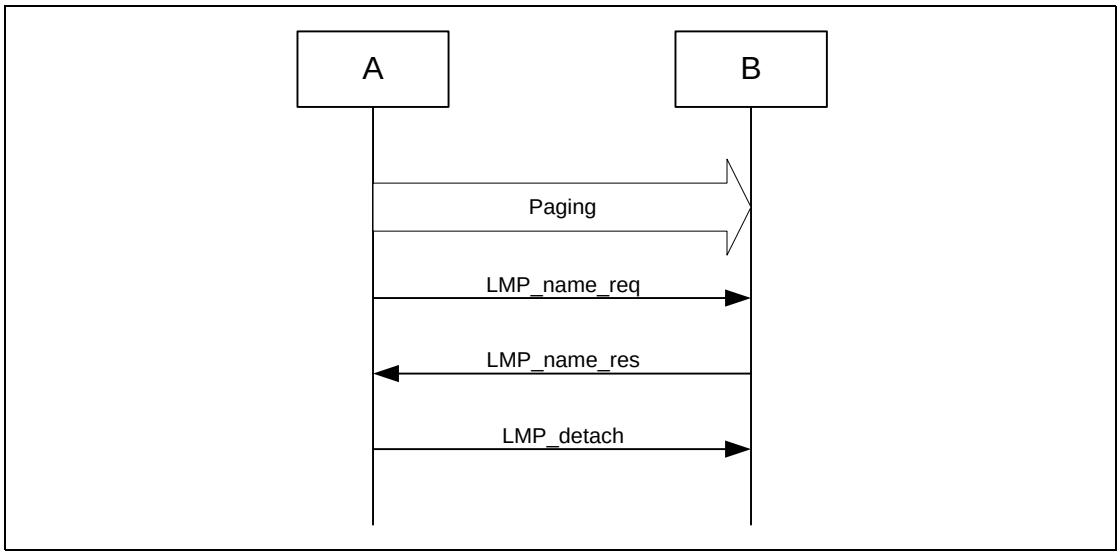



*Figure 6.3: Name request procedure.*

#### *6.3.3.2 Name discovery*

Name discovery is the procedure for retrieving the Bluetooth Device Name from connectable Bluetooth devices by performing name request towards known devices (i.e., Bluetooth devices for which the Bluetooth Device Addresses are available).

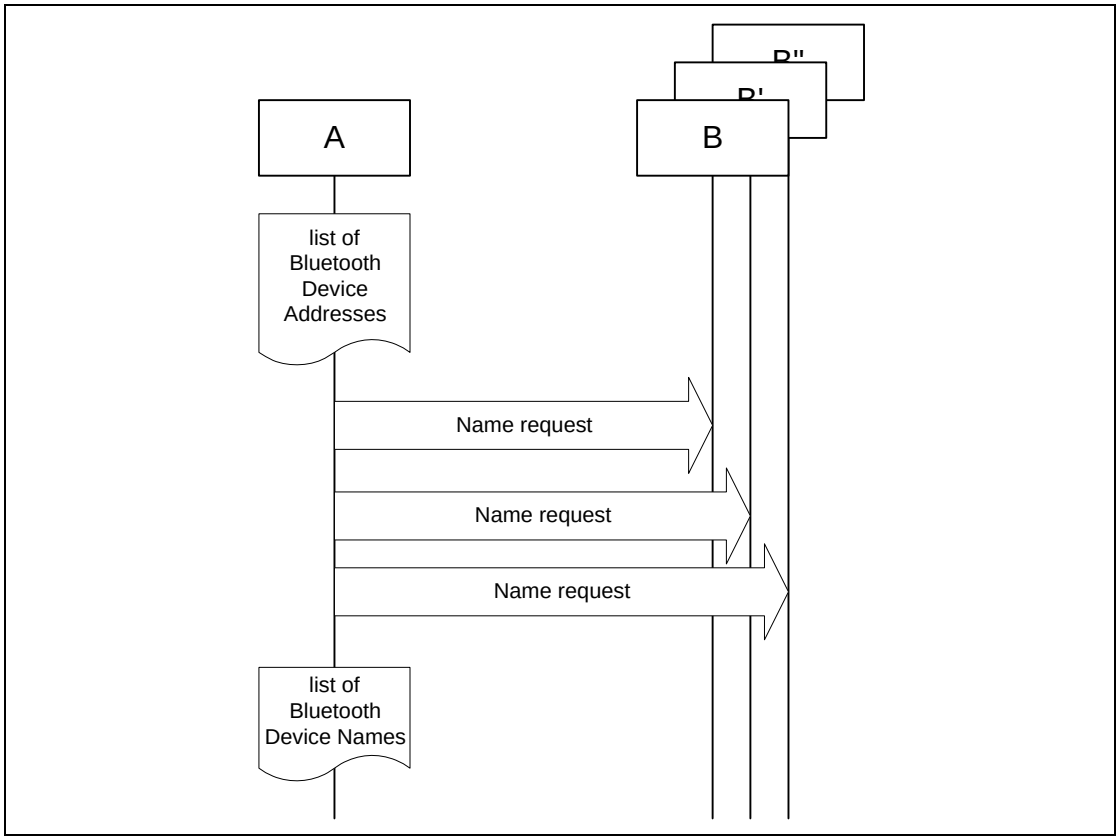



*Figure 6.4: Name discovery procedure.*

### 6.3.4 Conditions

In the name request procedure, the initiator will use the Device Access Code of the remote device as retrieved immediately beforehand – normally through an inquiry procedure.

## 6.4 DEVICE DISCOVERY

This section only applies to a device of the BR/EDR device type. Devices of the BR/EDR/LE device shall follow the device discovery defined in procedure in Section 13.2.3.

### 6.4.1 Purpose

The purpose of device discovery is to provide the initiator with the Bluetooth Device Address, clock, Class of Device, used page scan mode, Bluetooth Device Name, and extended inquiry response information of discoverable devices.

### 6.4.2 Term on UI Level

'Bluetooth Device Discovery'.

### 6.4.3 Description

During the device discovery procedure, first an inquiry (either general or limited) is performed, and then name discovery is done towards some or all of the devices that responded to the inquiry. If the initiator of the device discovery receives a complete local name or a shortened local name that is considered long enough, via an extended inquiry response from a remote device, the initiator should not do a separate name discovery for that device.

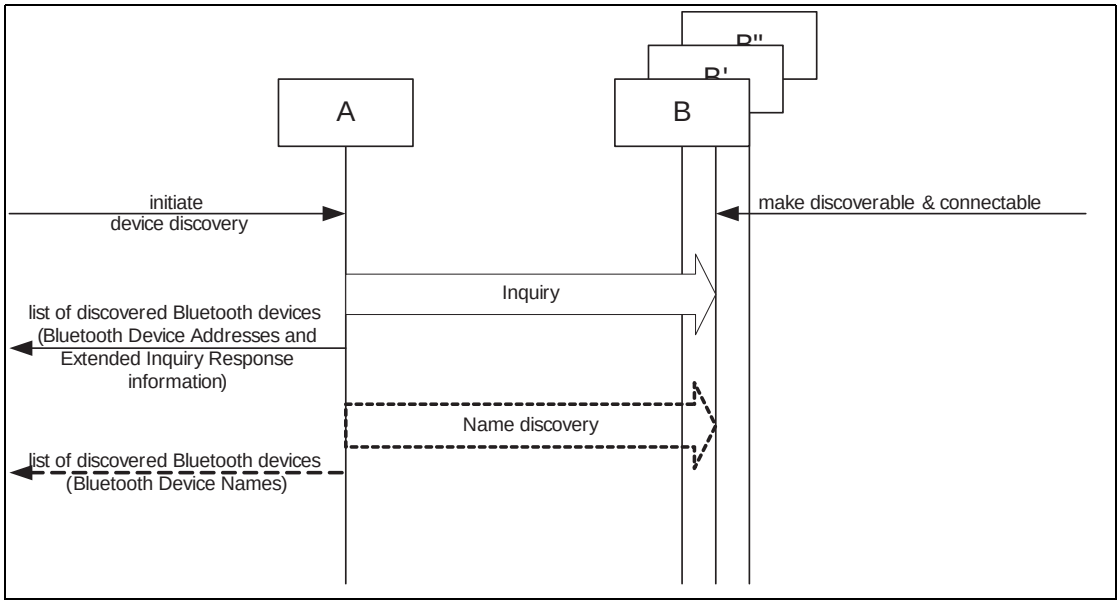



*Figure 6.5: Device discovery procedure.*

### 6.4.4 Conditions

Conditions for both inquiry (general or limited) and name discovery must be fulfilled (i.e., devices discovered during device discovery must be both discoverable and connectable).

## 6.5 BONDING

### 6.5.1 Purpose

The purpose of bonding is to create a relation between two Bluetooth devices based on a common link key (a bond). The link key is created and exchanged (pairing) during the bonding procedure and is expected to be stored by both Bluetooth devices, to be used for future authentication. In addition to pairing, the bonding procedure can involve higher-layer initialization procedures.

### 6.5.2 Term on UI level

'Bluetooth Bonding'.

### 6.5.3 Description

Two forms of bonding are described in the following sections: General Bonding and Dedicated Bonding.

#### *6.5.3.1 General Bonding*

General Bonding refers to the process of performing bonding during connection setup or channel establishment procedures as a precursor to accessing a service.

When the devices that are performing General Bonding both support Secure Simple Pairing, the Authentication_Requirements parameter should be set to MITM Protection Not Required – General Bonding unless the security policy of an available local service requires MITM Protection in which case the Authentication_Requirements parameter shall be set to MITM Protection Required – General Bonding. 'No bonding' is used when the device is performing a Secure Simple Pairing procedure, but does not intend to retain the link key after the physical link is disconnected.

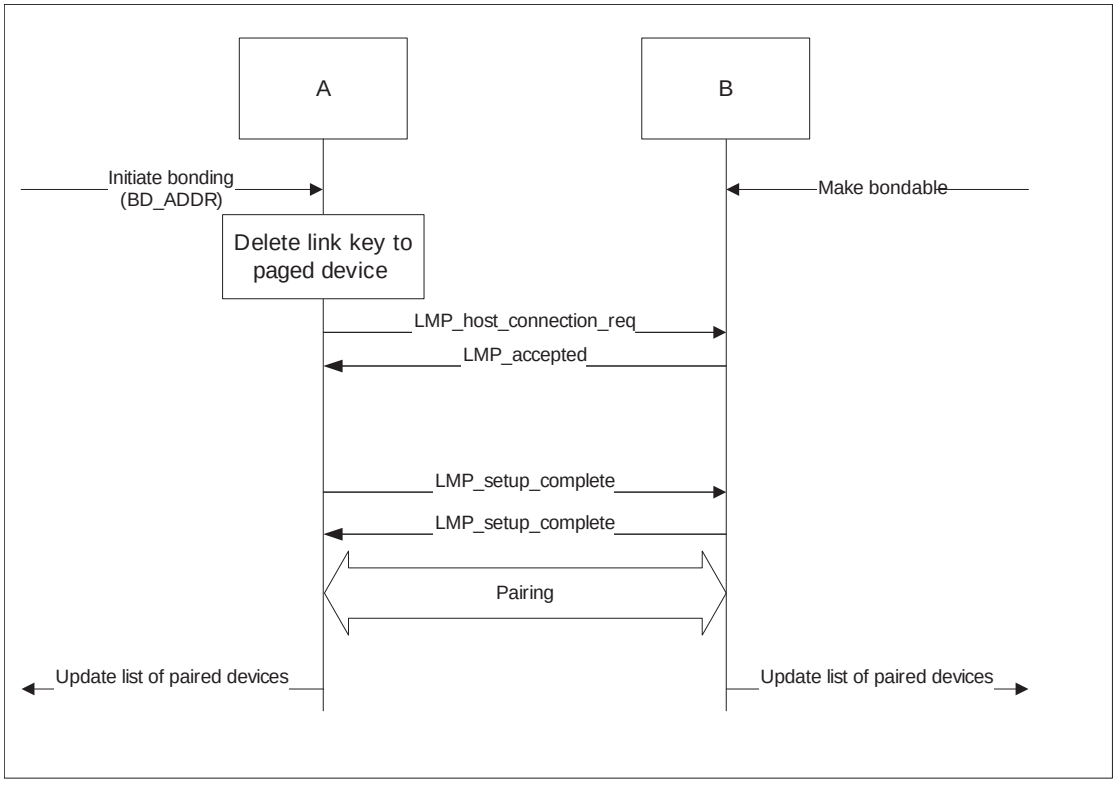



*Figure 6.6: General description of general bonding as being the link establishment procedure executed under specific conditions on both devices, followed by authentication and an optional higher layer initalization process.*

#### *6.5.3.2 Dedicated Bonding*

Dedicated Bonding refers to a procedure wherein one device connects to another only for the purpose of pairing without accessing a particular service.

The main difference with dedicated bonding, as compared to a pairing done during link or channel establishment, is that for bonding it is the paging device (A) that must initiate the authentication.

When the devices that are performing Dedicated Bonding both support Secure Simple Pairing, the Authentication_Requirements parameter should be set to *MITM Protection Not Required – Dedicated Bonding* unless the security policy of an available local service requires MITM Protection in which case the Authentication Required parameter shall be set to *MITM Protection Required – Dedicated Bonding*. 'No bonding' is used when the device is performing a Secure Simple Pairing procedure, but does not intend to retain the link key after the physical link is disconnected.

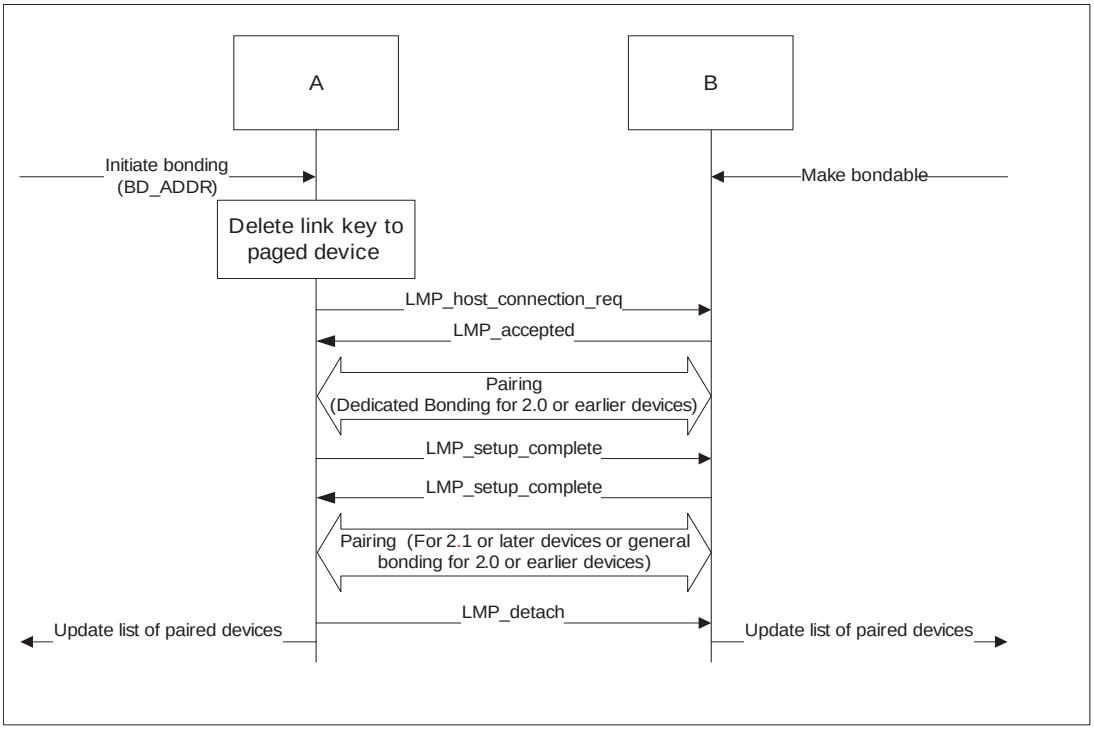



*Figure 6.7: Dedicated Bonding as performed when the purpose of the procedure is only to create and exchange a link key between two Bluetooth devices.*

### 6.5.4 Conditions

Before bonding can be initiated, the initiating device (A) must know the Device Access Code of the device to pair with. This is normally done by first performing device discovery. A Bluetooth Device that can initiate bonding (A) should use limited inquiry, and a Bluetooth Device that accepts bonding (B) should support the limited discoverable mode.

Bonding is in principle the same as link establishment with the conditions:

- The paged device (B) shall be set into bondable mode. The paging device (A) is assumed to allow pairing since it has initiated the bonding procedure.

- The paging device (the initiator of the bonding procedure, A) shall initiate authentication.
- Before initiating the authentication part of the bonding procedure, the paging device should delete any link key corresponding to a previous bonding with the paged device.

# 7 ESTABLISHMENT PROCEDURES

| | Procedure | Ref. | Support in A | Support in B |
|---|---|---|---|---|
| 1 | Link establishment | 7.1 | M | M |
| 2 | Channel establishment | 7.2 | O | M |
| 3 | Connection establishment | 7.3 | O | O |

*Table 7.1: Establishment procedures*

The establishment procedures defined here do not include any discovery part. Before establishment procedures are initiated, the information provided during device discovery (in the FHS packet or the extended inquiry response packet of the inquiry response or in the response to a name request) has to be available in the initiating device. This information is:

- The Bluetooth Device Address (BD_ADDR) from which the Device Access Code is generated;
- The system clock of the remote device;
- The page scan mode used by the remote device.

Additional information provided during device discovery that is useful for making the decision to initiate an establishment procedure is:

- The Class of device;
- The Device name;
- The supported Service Classes.

## 7.1 LINK ESTABLISHMENT

### 7.1.1 Purpose

The purpose of the link establishment procedure is to establish a physical link (of ACL type) between two Bluetooth devices using procedures from [1] and [2].

### 7.1.2 Term on UI Level

'Bluetooth link establishment'

### 7.1.3 Description

In this sub-section, the paging device (A) is in security mode 3. During link establishment, the paging device cannot distinguish if the paged device (B) is in security mode 1, 2 or 4.

### *7.1.3.1 B in security mode 1, 2, or 4*

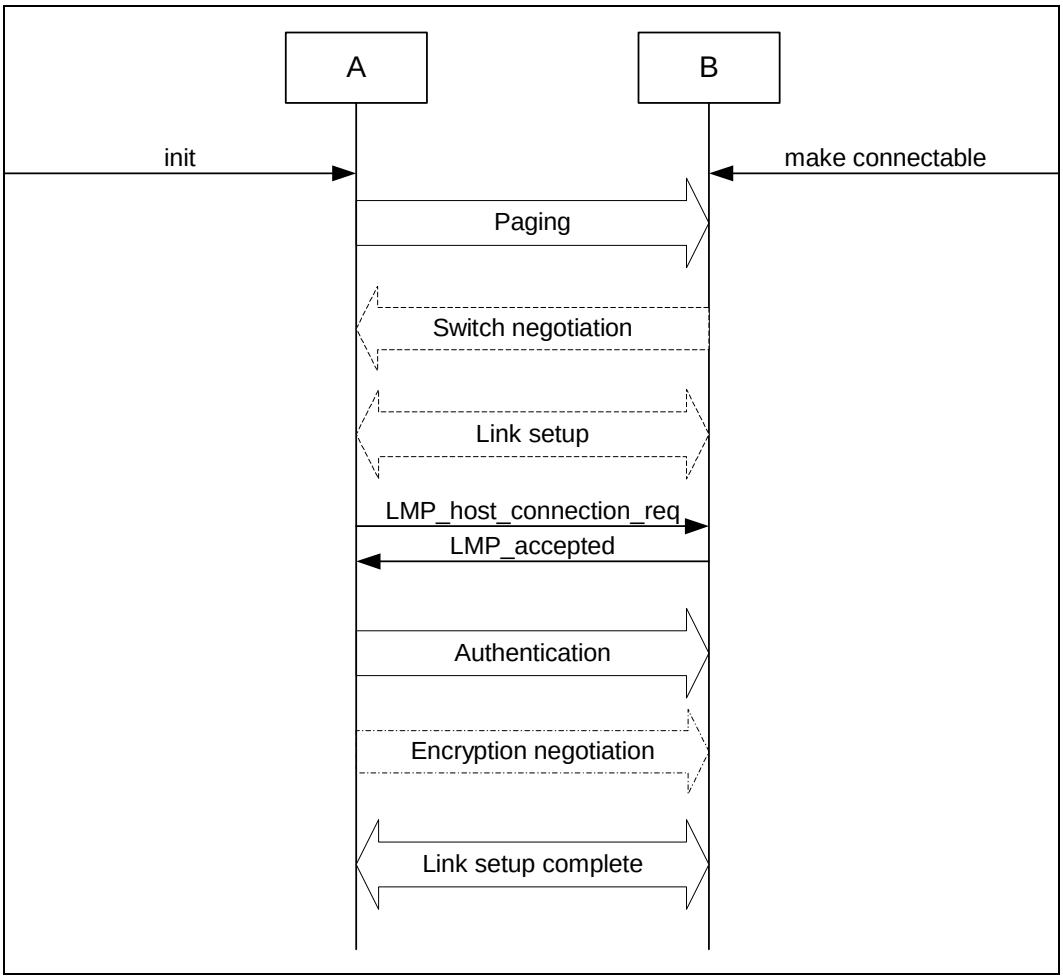



*Figure 7.1: Link establishment procedure when the paging device (A) is in security mode 3 and the paged device (B) is in security mode 1, 2, or 4.*

### *7.1.3.2 B in security mode 3*

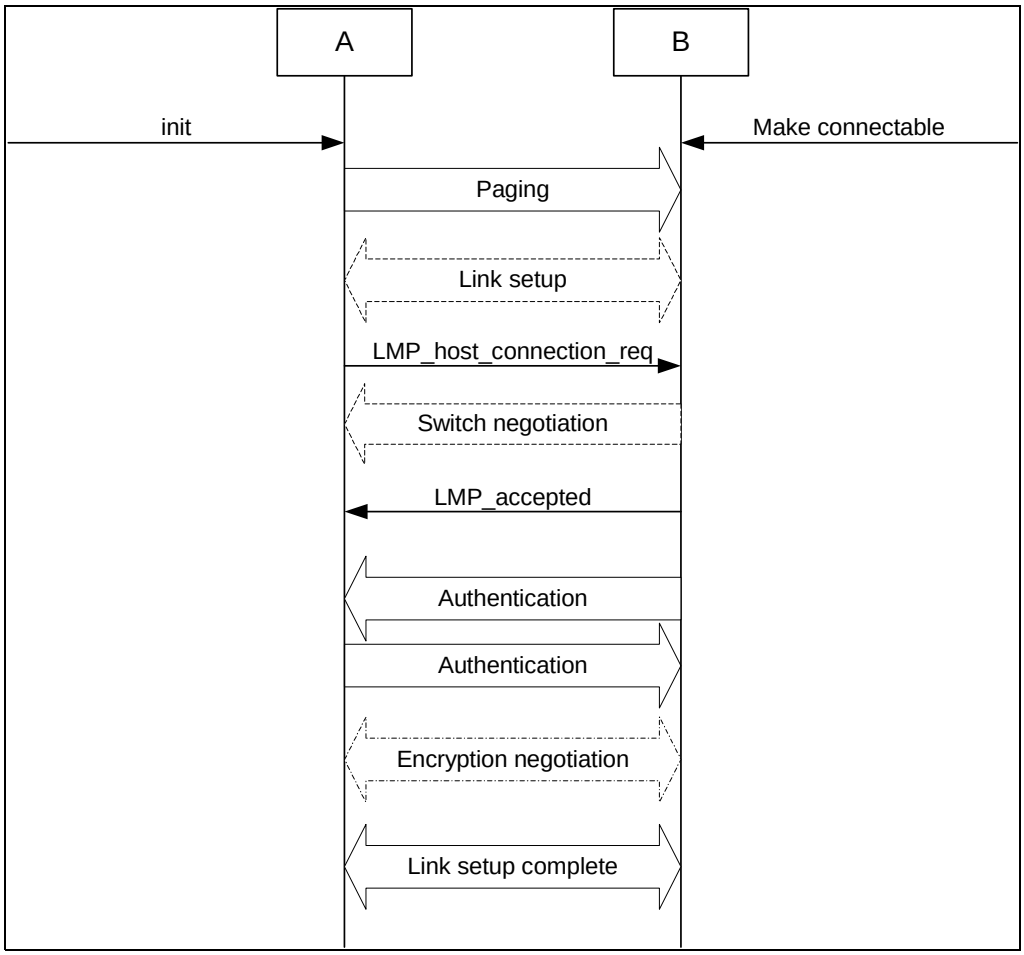



*Figure 7.2: Link establishment procedure when both the paging device (A) and the paged device (B) are in security mode 3.*

## 7.1.4 Conditions

The paging procedure shall be according to [Vol. 2], Part B Section 8.3 and the paging device should use the Device Access Code and page mode received through a previous inquiry. When paging is completed, a physical link between the two Bluetooth devices is established.

If role switching is needed (normally it is the paged device that has an interest in changing the master/slave roles) it should be done as early as possible after the physical link is established. If the paging device does not accept the switch, the paged device has to consider whether to keep the physical link or not.

Both devices may perform link setup (using LMP procedures that require no interaction with the host on the remote side). Optional LMP features can be used after having confirmed (using LMP_feature_req) that the other device supports the feature.

When the paging device needs to go beyond the link setup phase, it issues a request to be connected to the host of the remote device. If the paged device is in security mode 3, this is the trigger for initiating authentication.

The paging device shall send LMP_host_connection_req during link establishment (i.e., before channel establishment) and may initiate authentication only after having sent LMP_host_connection_req.

After an authentication has been performed, any of the devices can initiate encryption.

Further link configuration may take place after the LMP_host_connection_req. When both devices are satisfied, they send LMP_setup_complete.

Link establishment is completed when both devices have sent LMP_setup_complete.

## 7.2 CHANNEL ESTABLISHMENT

### 7.2.1 Purpose

The purpose of the channel establishment procedure is to establish a Bluetooth channel (L2CAP channel) between two Bluetooth devices using [1].

### 7.2.2 Term on UI level

'Bluetooth channel establishment'.

### 7.2.3 Description

In this sub-section, the initiator (A) is in security mode 3. During channel establishment, the initiator cannot distinguish if the acceptor (B) is in security mode 1 or 3.

#### *7.2.3.1 B in security mode 2 or 4*

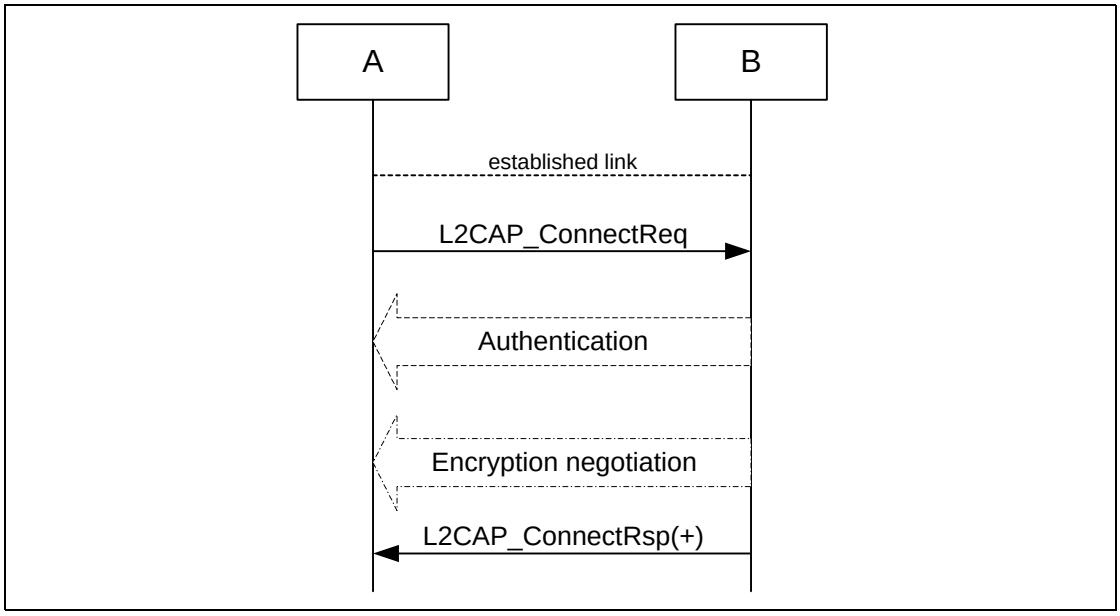



*Figure 7.3: Channel establishment procedure when the initiator (A) is in security mode 3 and the acceptor (B) is in security mode 2 or 4.*

#### *7.2.3.2 B in security mode 1 or 3*

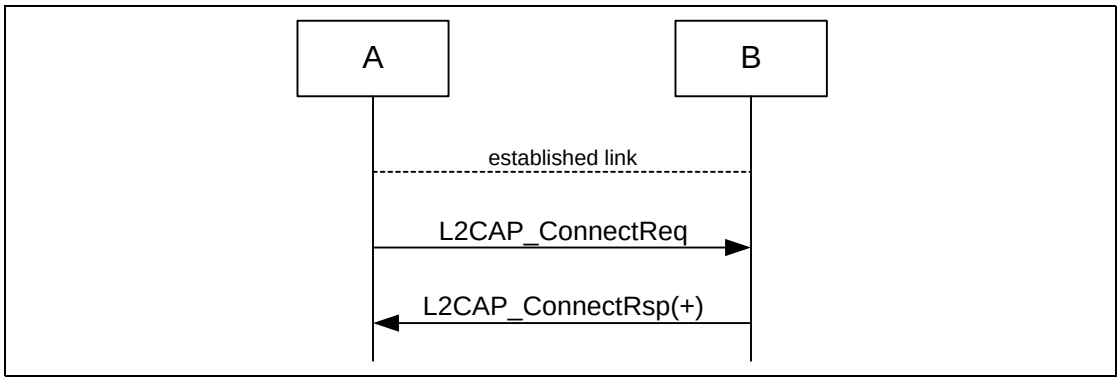



*Figure 7.4: Channel establishment procedure when the initiator (A) is in security mode 3 and the acceptor (B) is in security mode 1 or 3.*

### 7.2.4 Conditions

Channel establishment starts after link establishment is completed when the initiator sends a channel establishment request (L2CAP_ConnectReq).

Depending on security mode, security procedures may take place after the channel establishment has been initiated.

Channel establishment is completed when the acceptor responds to the channel establishment request (with a positive L2CAP_ConnectRsp).

## 7.3 CONNECTION ESTABLISHMENT

### 7.3.1 Purpose

The purpose of the connection establishment procedure is to establish a connection between applications on two Bluetooth devices.

### 7.3.2 Term on UI level

'Bluetooth connection establishment'

### 7.3.3 Description

In this sub-section, the initiator (A) is in security mode 3. During connection establishment, the initiator cannot distinguish if the acceptor (B) is in security mode 1 or 3.

*7.3.3.1 B in security mode 2 or 4*

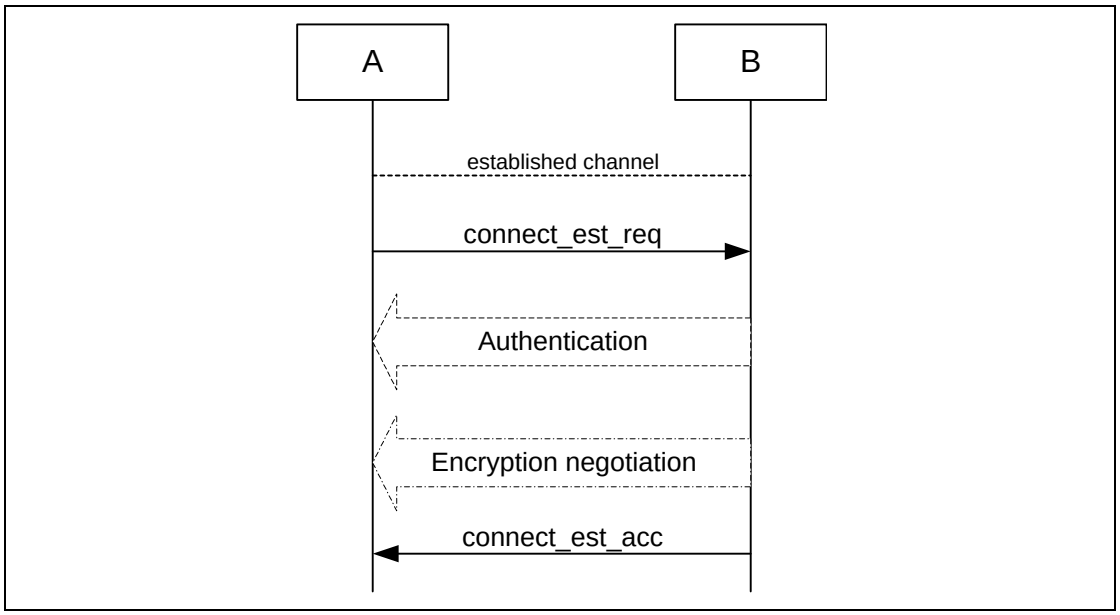



*Figure 7.5: Connection establishment procedure when the initiator (A) is in security mode 3 and the acceptor (B) is in security mode 2 or 4*

#### *7.3.3.2 B in security mode 1 or 3*

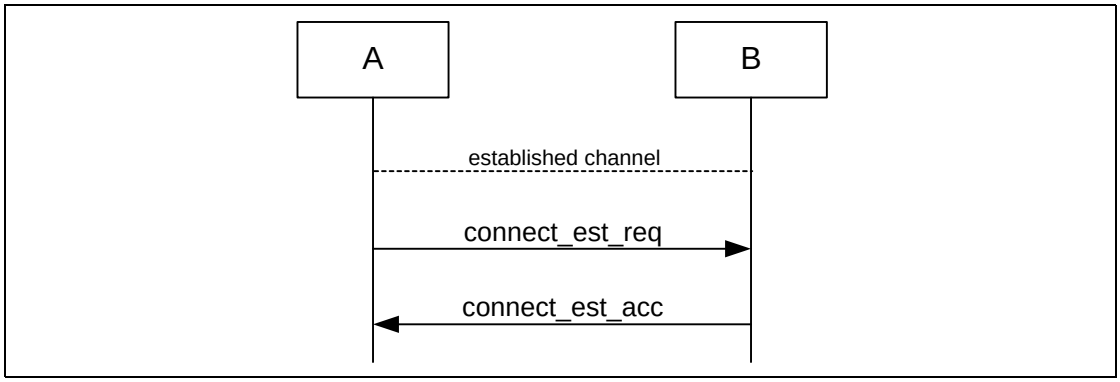



*Figure 7.6: Connection establishment procedure when the initiator (A) is in security mode 3 and the acceptor (B) is in security mode 1 or 3.*

### 7.3.4 Conditions

Connection establishment starts after channel establishment is completed, when the initiator sends a connection establishment request (’connect_est_req’ is application protocol-dependent). This request may be a TCS SETUP message [4] in the case of a Bluetooth telephony application Cordless Telephony Profile, or initialization of RFCOMM and establishment of DLC [3] in the case of a serial port-based application Serial Port Profile (although neither TCS or RFCOMM use the term ‘connection’ for this).

Connection establishment is completed when the acceptor accepts the connection establishment request (’connect_est_acc’ is application protocol dependent).

## 7.4 ESTABLISHMENT OF ADDITIONAL CONNECTION

When a Bluetooth device has established one connection with another Bluetooth device, it may be available for establishment of:

- A second connection on the same channel, and/or
- A second channel on the same logical link, and/or
- A second physical link.

If the new establishment procedure is to be towards the same device, the security part of the establishment depends on the security modes used. If the new establishment procedure is to be towards a new remote device, the device should behave according to active modes independent of the fact that it already has another physical link established (unless allowed co-incident radio and baseband events have to be handled).

# 8 EXTENDED INQUIRY RESPONSE DATA FORMAT

The extended inquiry response data format is shown in Figure 8.1. The data is 240 octets and consists of a significant part and a non-significant part. The significant part contains a sequence of data structures. Each data structure shall have a length field of one octet, which contains the Length value, and a data field of Length octets. The first n octets of the data field contain the extended inquiry response (EIR) data type. The content of the remaining Length - n octets in the data field depends on the value of the EIR data type and is called the EIR data. The non-significant part extends the extended inquiry response to 240 octets and shall contain all-zero octets.

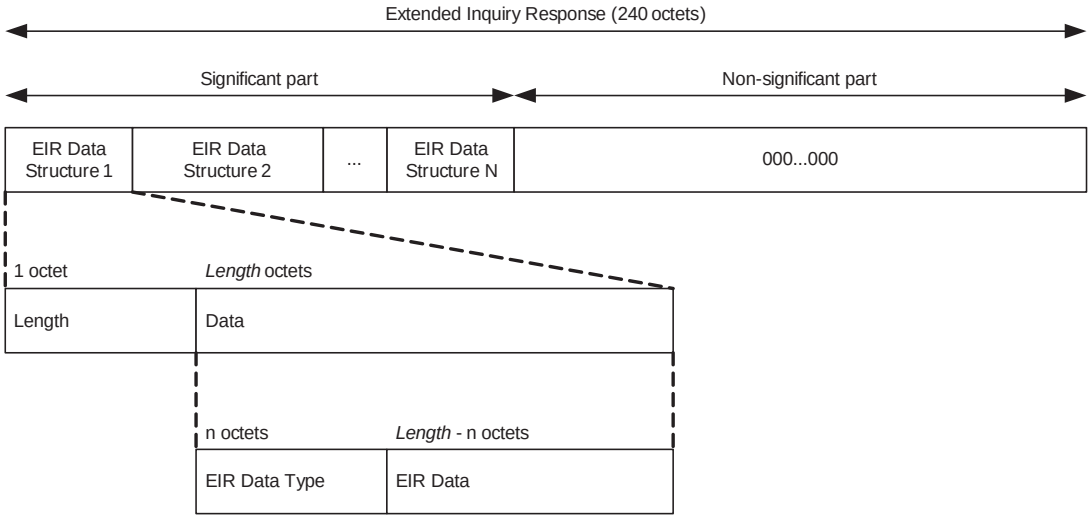



*Figure 8.1: Extended Inquiry Response data format*

The extended inquiry response data types are specified in the Assigned Numbers document.

If the length field is set to zero, then the data field has zero octets. This shall only occur to allow an early termination of the tagged data.

To reduce interference, the host should try to minimize the amount of EIR data such that the baseband can use a 1-slot or 3-slot EIR packet. This is advantageous because it reduces interference and maximizes the probability that the EIR packet will be received. If applications on a host provide more than 240 bytes of extended inquiry response data, it is up to the host to limit it to 240 octets.

The EIR data shall always be sent during inquiry response state. EIR data can contain device name, Tx power level, service class UUIDs, as well as manufacturers data, as defined in Section 8.1. In selecting the packet type to be used, FEC (DM1 or DM3) should be considered to maximize the range.

The Host shall include the device name in the EIR data according to the following rules:

1. If the device does not have a device name (i.e., 0 octet) and
    a) If there is no other data to be sent in the EIR packet, then host shall send a name tag with zero length and the type field set to indicate that this is the complete name (i.e., total of 2 octets with length = 1).
    b) If there is other important data to be sent in the EIR packet and a zero octet name tag will not fit, then host may avoid sending the name tag.
2. If the device has a device name (greater than 0 octet) and
    a) If it is too long be included in the EIR packet (given the choice of packet type and any other data that is being sent), then host may send a shortened version of the name (even 0 octet) and shall mark the name as 'shortened' to inform the receiver that a remote name request is required obtain the full name if the name is needed.
    b) If there are no other data to be sent in the EIR packet (given the choice of packet type selected), then host shall maximize the length of the device name to be sent, this may be complete or shortened name (e.g., if DM1 packet is chosen and device name characters equates to greater than 15 octets, then host sends first few characters that equates to 15 octets or less with shortened flag).

Note: It is not necessary to understand each and every EIR data type. If a host does not understand a given EIR data type value it should just skip over Length octets and look for the next EIR data structure.

## 8.1 EIR DATA TYPE DEFINITIONS

This section defines the basic EIR data types. Additional EIR data types may be defined in profile specifications.

All EIR data type values are listed in the Bluetooth Assigned Numbers document.

All numerical multi-byte entities and values associated with the following data types shall use little-endian byte order.

### 8.1.1 Service Class UUIDs

A list of service class UUIDs (see Section 2.4, “Service Class,” on page 215) may be included in the extended inquiry response data packet. There are three types of UUIDs that may be returned: 16-bit Bluetooth UUIDs, 32-bit Bluetooth UUIDs, and global 128-bit UUIDs. Each size is assigned two data type values.

In addition to the size, the data type values also indicate if the list of UUIDs is complete or there are more UUIDs available via SDP. An extended inquiry response shall not contain more than one list for each size of UUIDs. If a device has no service class UUIDs of a certain size the corresponding field in the extended inquiry response shall be marked as complete with no service class UUIDs. An omitted service class UUID list shall be interpreted as an empty incomplete list.

The format of the data is defined in Section 18.2.

### 8.1.2 Local Name

When included in the extended inquiry response, this shall be the same as, or a shortened version of, the local name assigned to the device. The data type value indicates if the name is complete or shortened. If the name is shortened, the complete name can be read using the remote name request procedure. An extended inquiry response shall not contain more than one instance of this data type.

The format of the data is defined in Section 18.3.

### 8.1.3 Flags

This data type is allocated for flag bits. These may for example be used for boolean information that would otherwise require a whole byte (not counting the Length and Data Type fields). The flag bits shall be included in the extended inquiry response if any of the bits are non-zero. If all flag bits are zero they may be omitted. An extended inquiry response shall not contain more than one instance of this data type.

The format of the data is defined in Section 18.1.

The LE Limited Discoverable Mode and LE General Discoverable Mode flags shall be ignored when received over the BR/EDR physical channel. The ‘BR/EDR Not Supported’ flag shall not be sent over the BR/EDR physical channel.

### 8.1.4 Manufacturer Specific Data

This data type is used for manufacturer specific data. The first two data octets shall contain a company identifier code from the Assigned Numbers - Company Identifiers document. The interpretation of any other octets within the data shall be defined by the manufacturer specified by the company identifier.The format of the data is defined in Section 18.11.

| Value | Description |
|---|---|
| 0xFF | Manufacturer Specific Data |

### 8.1.5 TX Power Level

The TX Power Level tag indicates the transmitted power level of the EIR packet.

The format of the data is defined in Section 18.4.

The TX Power Level tag may be used to calculate path loss on a received EIR packet using the following equation:

pathloss = Tx Power Level – EIR_RSSI

For example, if Tx Power Level = +4 (dBm) and the RSSI on the EIR packet is -60 (dBm) then the total pathloss is +4 – (-60) = +64 dB. If a second EIR packet were received at -40dBm with a Tx Power Level tag = +15dBm the resulting pathloss would be +55dB. An application could use these pathloss values to choose which device it thinks might be closer (the one with the lower pathloss value).

Unfortunately, due to fading and varying antenna, circuit, and chip characteristics, these resulting pathloss values will have uncertainty. Some of the uncertainty (for example, due to fading) may be able to be removed if multiple EIR packets are received from the same device.

### 8.1.6 Secure Simple Pairing Out of Band (OOB)

An out of band mechanism may be used in Secure Simple Pairing to communicate discovery information as well as other information related to the pairing process. These tags use the standard EIR tag format. They shall only be used over an out-of-band mechanism.

| Value | Description |
|---|---|
| NA | Length (2 bytes)<br>0xXXXX: 8 to 65535 bytes<br>This field contains the length of the entire OOB data block including the length field itself. |
| NA | BD_ADDR (6 bytes)<br>Format defined in [Vol. 2, Part B] Section 1.2 on page 68 |

*Table 8.1: Mandatory OOB Tags*

The format of the Optional OOB Tags is defined in Section 18.5.

## 8.2 EXAMPLE EXTENDED INQUIRY RESPONSE

This is an example extended inquiry response for a phone with PANU, Hands-free Audio Gateway, and Intercom Service:

| Value | Notes |
|---|---|

| | |
|---|---|
| 0x06 | Length of this Data |
| 0x09 | Complete Local Name |
| 0x50 | 'P' |
| 0x68 | 'h' |
| 0x6F | 'o' |
| 0x6E | 'n' |
| 0x65 | 'e' |
| 0x07 | Length of this Data |
| 0x03 | Complete list of 16-bit Service Class UUIDs |
| 0x15 | PANU service class UUID |
| 0x11 | |
| 0x1F | Handsfree Audio Gateway service class UUID |
| 0x11 | |
| 0x09 | Cordless Telephony service class UUID |
| 0x11 | |
| 0x01 | Length of this data |
| 0x05 | Complete list of 32-bit Service Class UUIDs |
| 0x01 | Length of this data |
| 0x07 | Complete list of 128-bit Service Class UUIDs |
| 0x00 | End of Data (Not transmitted over the air) |

# 9 OPERATIONAL MODES AND PROCEDURES FOR USE ON LE PHYSICAL CHANNELS

Several different modes and procedures may be performed simultaneously over an LE physical channel. The following modes and procedures are defined for use over LE physical channels:

- Broadcast mode and observation procedure
- Discovery modes and procedures
- Connection modes and procedures
- Bonding modes and procedures

Each of the above modes and procedures are independent from each other but are closely related since a combination of the modes and procedures are necessary for most devices to communicate with each other. Both the modes and procedures may be entered or executed respectively as a result of direct user action or autonomously by a device.

The Host shall configure the Controller with its local Link Layer feature information as defined in [Vol. 6], Part B Section 4.6 before performing any of the above modes and procedures.

## 9.1 BROADCAST MODE AND OBSERVATION PROCEDURE

The broadcast mode and observation procedure allow two devices to communicate in a unidirectional connectionless manner using the advertising events. The requirements for a device operating in a specific GAP role to support the broadcast mode and observation procedure are defined in Table 9.1.

| Broadcast Mode and Observation procedure | Ref. | Broadcaster | Observer |
|---|---|---|---|
| Broadcast mode | 9.1.1 | M | E |
| Observation procedure | 9.1.2 | E | M |

*Table 9.1: Broadcast mode and Observation procedure Requirements*

### 9.1.1 Broadcast Mode

#### *9.1.1.1 Definition*

The broadcast mode provides a method for a device to send connectionless data in advertising events.

#### *9.1.1.2 Conditions*

A device in the broadcast mode shall send data in either non-connectable undirected or scannable undirected advertising events.

The advertising data shall be formatted using the Advertising Data (AD) type format as defined in Section 11. A device in the broadcast mode shall not set the 'LE General Discoverable Mode' flag or the 'LE Limited Discoverable Mode' flag in the Flags AD Type as defined in Section 11.1.3.

When privacy is enabled, as defined in Section 10.7, and the device is operating in the broadcast mode, the Host shall generate a resolvable private address using the resolvable private address generation procedure as defined in Section 10.8.2.2. The Host shall configure the Controller to use the resolvable private address as the advertiser's device address in the advertising events. If privacy is disabled then the Host shall disable the $T_{GAP}$(private_addr_int) timer and configure the Controller to use the public/static address as the advertiser's device address in the advertising events. When the $T_{GAP}$(private_addr_int) timer expires the Host shall repeat the above procedure. The device should not send the device name in the advertising data. If the device receives a scan request, the device shall only respond if the scan request is from a known device.

Note: All data sent by a device in the broadcast mode is considered unreliable since there is no acknowledgement from any device that may have received the data.

Note: A device that only supports the broadcast mode cannot support sending encrypted or signed data because the device cannot support the distribution of keys as defined by the Security Manager (see [Vol. 3], Part H). Key distribution may be performed by a device using the Peripheral or Central profile role.

### 9.1.2 Observation Procedure

#### *9.1.2.1 Definition*

The observation procedure provides a method for a device to receive connectionless data from a device that is sending advertising events.

#### *9.1.2.2 Conditions*

A device performing the observation procedure may use passive scanning or active scanning to receive advertising events.

A device performing the observation procedure may use active scanning to also receive scan response data sent by any device in the broadcast mode that advertises using scannable undirected advertising events. When a device performing the observation procedure receives a resolvable private address in the

advertising event, the device may resolve the private address by using the resolvable private address resolution procedure as defined in Section 10.8.2.3.

A privacy enabled device as defined in Section 10.7 performing the observation procedure shall use a non-resolvable private address as the device address when performing active scanning. The generation of a non-resolvable address is defined in Section 10.8.2.1.

Note: In use cases where a device in the broadcast mode sends dynamic data, the receiving device should disable duplicate filtering capability in the Controller so that the Host receives all advertising packets received by the Controller.

Note: A device performing the observation procedure cannot support receiving of encrypted or signed data because the device cannot support the distribution of keys as defined by Security Manager in [Vol. 3], Part H Section 3.6.1. Key distribution may be performed by a device using the Peripheral or Central profile role.

## 9.2 DISCOVERY MODES AND PROCEDURES

All devices shall be in either non-discoverable mode or one of the discoverable modes. A device in the discoverable mode shall be in either the general discoverable mode or the limited discoverable mode. A device in the non-discoverable mode is not discoverable. Devices operating in either the general discoverable mode or the limited discoverable mode can be found by the discovering device. A device that is discovering other devices performs either the limited discovery procedure as defined in Section 9.2.5 or the general discovery procedure as defined in Section 9.2.6.

### 9.2.1 Requirements

| Discovery modes and procedures | Ref. | Peripheral | Central |
|---|---|---|---|
| Non-Discoverable mode | 9.2.2 | M | E |
| Limited Discoverable mode | 9.2.3 | O | E |
| General Discoverable mode | 9.2.4 | C1 | E |
| Limited Discovery procedure | 9.2.5 | E | O |
| General Discovery procedure | 9.2.6 | E | M |
| Name Discovery procedure | 9.2.7 | O | O |
| C1: if limited discoverable mode is not supported then general discoverable mode is mandatory, else optional. | | | |

*Table 9.2: Device discovery requirements*

### 9.2.2 Non-Discoverable Mode

#### *9.2.2.1 Description*

A device configured in non-discoverable mode will not be discovered by any device that is performing either the general discovery procedure or the limited discovery procedure.

#### *9.2.2.2 Conditions*

A device in the non-discoverable mode that sends advertising events shall not set the ‘LE General Discoverable Mode’ flag or ‘LE Limited Discoverable Mode’ flag in the Flags AD type. A Peripheral device in the non-connectable mode may send non-connectable undirected advertising events or scannable undirected advertising events or may not send advertising packets.

### 9.2.3 Limited Discoverable Mode

#### *9.2.3.1 Description*

Devices configured in the limited discoverable mode are discoverable for a limited period of time by other devices performing the limited or general device discovery procedure. The limited discoverable mode is typically used when a user performs a specific action to make the device discoverable for a limited period of time.

#### *9.2.3.2 Conditions*

While a device is in the Peripheral role the device may support the limited discoverable mode. While a device is in the Broadcaster, Observer or Central role the device shall not support the limited discoverable mode.

A device in the limited discoverable mode sends either non-connectable advertising events, scannable undirected advertising events or connectable undirected advertising events.

While in the limited discoverable mode the device shall send advertising event types with the advertising data including the Flags AD type as defined in Section 11.1.3 with the following flags set as described:

- The LE Limited Discoverable Mode flag shall be set to one
- The LE General Discoverable Mode flag shall be set to zero.
- A device of the LE-only device type shall set the 'BR/EDR Not Supported' flag to one, the 'Simultaneous LE and BR/EDR to Same Device Capable (Controller)' flag to zero, and the 'Simultaneous LE and BR/EDR to Same Device Capable (Host)' flag to zero

The advertising data should also include the following AD types to enable a faster connectivity experience:

- TX Power Level AD type defined in Section 11.1.5.
- Local Name AD type defined in Section 11.1.2.
- Service UUIDs AD type defined in Section 11.1.1.
- Slave Connection Interval Range AD type as defined in Section 11.1.7.

Devices shall remain in the limited discoverable mode no longer than $T_{GAP}$(lim_adv_timeout).

While a device is in limited discoverable mode the Host configures the Controller as follows:

- The Host shall set the advertising filter policy to 'process scan and connection requests from all devices'
- The Host should set the minimum advertising interval to $T_{GAP}$(lim_disc_adv_int_min)
- The Host should set the maximum advertising interval to $T_{GAP}$(lim_disc_adv_int_max).

The device shall remain in limited discoverable mode until a connection is established or the Host terminates the mode.

If a device is in the limited discoverable mode and in the undirected connectable mode the advertising interval values should be set as defined in Section 9.3.4

A device in the limited discoverable mode shall not set both the LE Limited Discoverable Flag and the LE General Discoverable Flag to one.

Note: Data that change frequently should be placed in the advertising data and static data should be placed in the scan response data.

Note: The choice of advertising interval is a trade-off between power consumption and device discovery time.

### 9.2.4 General Discoverable Mode

#### *9.2.4.1 Description*

Devices configured in general discoverable mode are intended to be discoverable by devices performing the general discovery procedure. The general discoverable mode is typically used when the device is intending to be discoverable for a long period of time.

#### *9.2.4.2 Conditions*

While a device is in the Peripheral role the device may support the general discoverable mode. While a device is in the Broadcaster, Observer or Central role the device shall not support the general discoverable mode.

A device in the general discoverable mode sends either non-connectable advertising events, scannable undirected advertising events, or connectable undirected advertising events.

While in general discoverable mode the device shall send advertising events with the advertising data including the Flags AD data type as defined in Section 11.1.3 with the following flags:

- The LE Limited Discoverable Mode flag set to zero
- The LE General Discoverable Mode flag set to one
- A device of the LE-only device type shall set the ‘BR/EDR Not Supported’ flag to one, the ‘Simultaneous LE and BR/EDR to Same Device Capable (Controller)’ flag to zero, and the ‘Simultaneous LE and BR/EDR to Same Device Capable (Host)’ flag to zero

The advertising data should also include the following AD types to enable a faster connectivity experience:

- TX Power Level AD type as defined in Section 11.1.5.
- Local Name AD type as defined in Section 2.
- Service UUIDs AD type as defined in Section 11.1.1**.**
- Slave Connection Interval Range AD type as defined in Section 11.1.7.

While a device is in general discoverable mode the Host configures the Controller as follows:

- The Host shall set the advertising filter policy to ‘process scan and connection requests from all devices’
- The Host should set the minimum advertising interval to $T_{GAP}$(gen_disc_adv_int_min)

- The Host should set the maximum advertising interval to $T_{GAP}$(gen_disc_adv_int_max).

The device shall remain in general discoverable mode until a connection is established or the Host terminates the mode.

If a device is in the general discoverable mode and in the directed connectable mode or the non-connectable mode, the advertising interval values should be set as defined in Section 9.3.3.

A device in the general discoverable mode shall not set both the LE Limited Discoverable Flag and the LE General Discoverable Flag to one.

Note: Data that change frequently should be placed in the advertising data and static data should be placed in the scan response data.

Note: The choice of advertising interval is a trade-off between power consumption and device discovery time.

### 9.2.5 Limited Discovery Procedure

#### *9.2.5.1 Description*

A device performing the limited discovery procedure receives the device address, advertising data and scan response data from devices in the limited discoverable mode only.

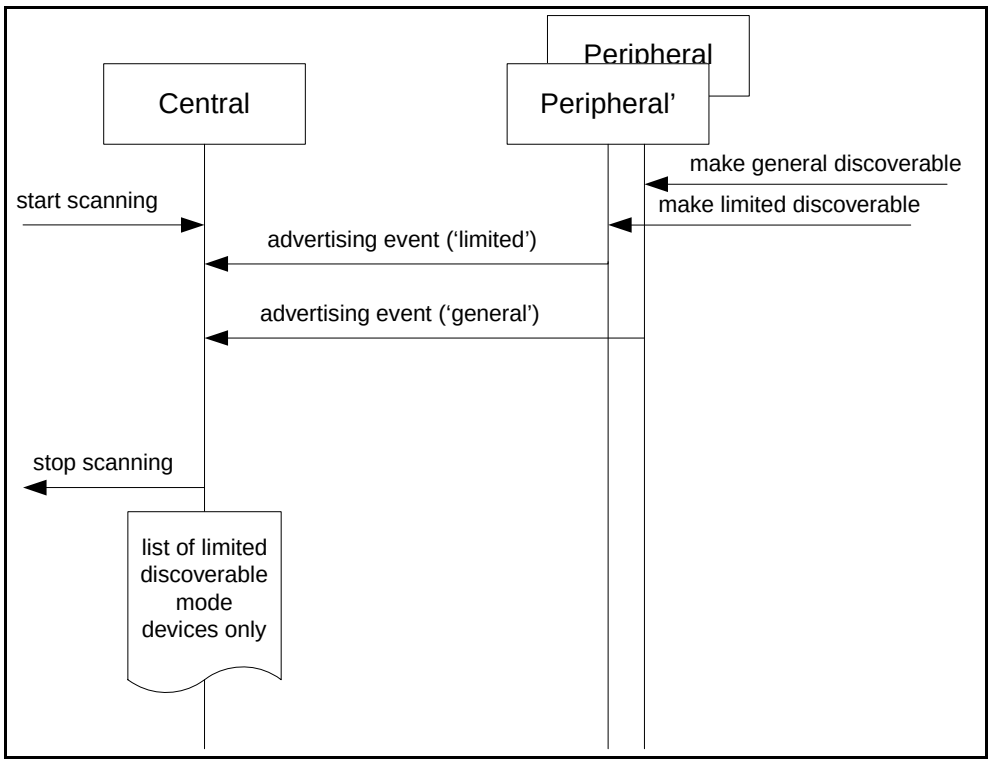



*Figure 9.1: A Central performing limited discovery procedure discovering Peripherals in the limited discoverable mode*

### *9.2.5.2 Conditions*

While a device is in the Central role the device may support the limited discovery procedure. While a device is in the Broadcaster, Observer or Peripheral role the device shall not support the limited discovery procedure.

When a Host performs the limited discovery procedure, the Host configures the Controller as follows:

1. The Host shall set the scanner filter policy to 'process all advertising packets'
2. The Host should set the scan interval of $T_{GAP}$(lim_disc_scan_int)
3. The Host should set the scan window of $T_{GAP}$(lim_disc_scan_wind)
4. The Host should configure the Controller to use active scanning.

The Host shall begin scanning for advertising packets and should continue for a minimum of $T_{GAP}$(lim_disc_scan_min), unless the host ends the limited discovery procedure.

The Host shall check for the Flags AD type in the advertising data. If the Flags AD type is present and the LE Limited Discoverable Flag is set to one then the Host shall consider the device as a discovered device, otherwise the advertising data shall be ignored. The Flag AD type is defined in Section 11.1.3. The advertising data of the discovered device may contain data with other AD

types, e.g. Service UUIDs AD type, TX Power Level AD type, Local Name AD type, Slave Connection Interval Range AD type. The Host may use the data in performing any of the connection establishment procedures.

### 9.2.6 General Discovery Procedure

#### *9.2.6.1 Description*

A device performing the general discovery procedure receives the device address, advertising data and scan response data from devices in the limited discoverable mode or the general discoverable mode.

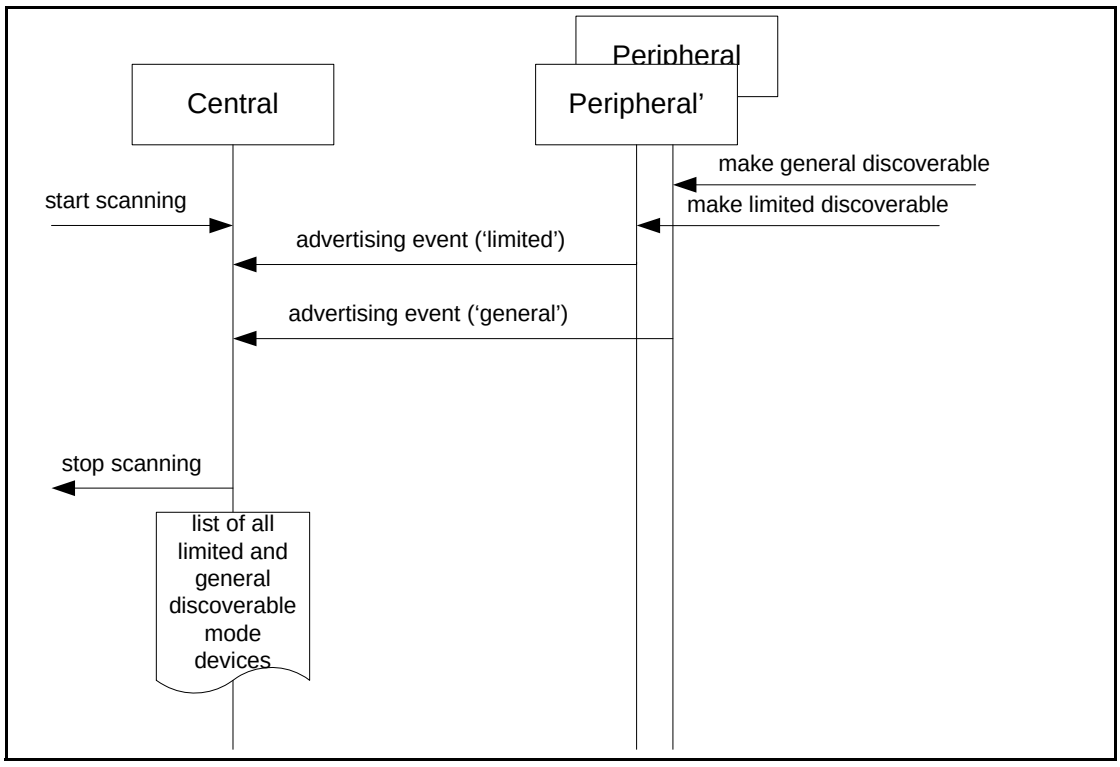



*Figure 9.2: A Central performing general discovery procedure discovering Peripherals in the limited discoverable mode and general discoverable mode.*

#### *9.2.6.2 Conditions*

While a device is in the Central role the device shall support the general discovery procedure. While a device is in the Broadcaster, Observer or Peripheral role the device shall not support the general discovery procedure.

When a Host performs the general discovery procedure, the Host configures the Controller as follows:

1. The Host shall set the scanner filter policy to 'process all advertising packets'
2. The Host should set the scan interval of $T_{GAP}$(gen_disc_scan_int)

3. The Host should set the scan window of $T_{GAP}$(gen_disc_scan_wind)
4. The Host should configure the Controller to use active scanning

The Host shall begin scanning for advertising packets and should continue for a minimum of $T_{GAP}$(gen_disc_scan_min). The procedure may be terminated early by the Host.

The Host shall check for the Flags AD type in the advertising data. If the Flags AD type is present and either the LE General Discoverable Mode flag is set to one or the LE Limited Discoverable Mode flag is set to one then the Host shall consider the device as a discovered device, otherwise the advertising data shall be ignored. The advertising data of the discovered device may contain data with other AD types, e.g. Service UUIDs AD type, TX Power Level AD type, Local Name AD type, Slave Connection Interval Range AD type. The Host may use the data in performing any of the connection establishment procedures as defined in Section 9.3.

### 9.2.7 Name Discovery Procedure

#### *9.2.7.1 Description*

The name discovery procedure is used to obtain the Bluetooth Device Name of a remote connectable device.

#### *9.2.7.2 Conditions*

If the complete device name is not acquired while performing either the limited discovery procedure or the general discovery procedure, then the name discovery procedure may be performed.

The name discovery procedure shall be performed as follows:

1. The Host shall establish a connection using one of the connection establishment procedures as defined in Section 9.3.
2. The Host shall read the device name characteristic using the GATT procedure Read Using Characteristic UUID [Vol. 3], Part G Section 4.8.2
3. The connection may be terminated after the GATT procedure has completed.

## 9.3 CONNECTION MODES AND PROCEDURES

The connection modes and procedures allow a device to establish a connection to another device.

A Peripheral device may support the privacy feature as defined in Section 10.7 in the directed and undirected modes. A Central may connect to a Peripheral that supports the privacy feature using only the general connection establishment procedure.

When devices are connected, the parameters of the connection can be updated with the Connection Parameter Update procedure. The connected device may terminate the connection using the Terminate Connection procedure. The requirements for a device to support the connection modes and procedures are defined in Table 9.3.

### 9.3.1 Requirements

| Connection Modes and Procedures | Ref. | Peripheral | Central | Broadcaster | Observer |
|---|---|---|---|---|---|
| Non-connect-able mode | 9.3.2 | M | E | M | M |
| Directed con-nectable mode | 9.3.3 | O | E | E | E |
| Undirected con-nectable mode | 9.3.4 | M | E | E | E |
| Auto connection establishment procedure | 9.3.5 | E | O | E | E |
| General connec-tion establish-ment procedure | 9.3.6 | E | C1 | E | E |
| Selective con-nection estab-lishment procedure | 9.3.7 | E | O | E | E |
| Direct connec-tion establish-ment procedure | 9.3.8 | E | M | E | E |
| Connection parameter update proce-dure | 9.3.9 | O | M | E | E |
| Terminate con-nection proce-dure | 9.3.10 | M | M | E | E |
| C1: Mandatory if privacy feature is supported, else optional | | | | | |

*Table 9.3: Connection modes and procedures requirements*

### 9.3.2 Non-Connectable Mode

#### *9.3.2.1 Description*

A device in the non-connectable mode shall not allow a connection to be established.

#### *9.3.2.2 Conditions*

While a device is in the Peripheral role the device shall support the non-connectable mode. A device in the Central role cannot send advertising events, therefore the device is considered to support the non-connectable mode. A

device in the Broadcaster or Observer role cannot establish a connection, therefore the device is considered to support the non-connectable mode.

A Peripheral device in the non-connectable mode may send non-connectable undirected advertising events or scannable undirected advertising events.

### 9.3.3 Directed Connectable Mode

#### *9.3.3.1 Description*

A device in the directed connectable mode shall accept a connection request from a known peer device performing the auto connection establishment procedure or the general connection establishment procedure.

#### *9.3.3.2 Conditions*

While a device is in the Peripheral role the device may support the direct connectable mode. This mode shall only be used if the device has a known peer device address. While a device is in the Broadcaster, Observer, or the Central role the device shall not support the direct connectable mode.

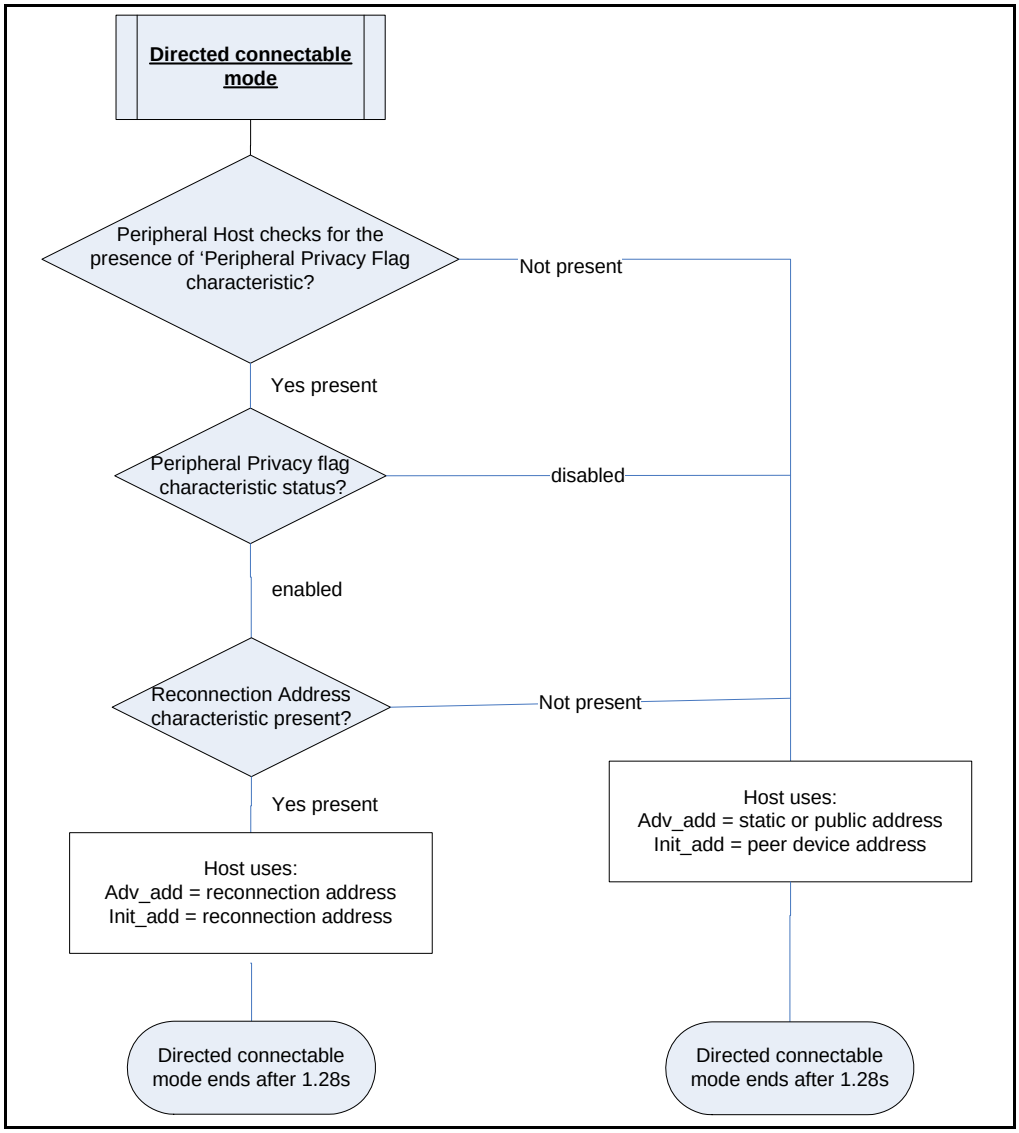



*Figure 9.3: Flow chart for a device entering the directed connectable mode*

A Host entering the directed connectable Mode shall perform the following steps:

1. If the Peripheral Privacy Flag characteristic exists on the local device, the Host shall proceed to step 2, otherwise the Host shall proceed to step 5.
2. If the value in the Peripheral Privacy Flag characteristic is set to 1 (privacy enabled), the Host shall proceed to step 3, otherwise the Host shall proceed to step 5.
3. If the Reconnection Address characteristic exists on the local device, the Host shall proceed to step 4, otherwise the Host shall proceed to step 5.
4. The Host uses the address in the Reconnection Address characteristic as the initiator's device address and the advertiser device address in the directed connectable advertising events. The Host shall proceed to step 6.

5. The Host uses the static or the public address of this device as the advertiser's device address. The Host uses the known peer device address as the initiator's device address. The Host shall proceed to step 6.
6. The Host should configure the Controller with the minimum advertising interval set to $T_{GAP}$(dir_conn_adv_int_min) and the maximum advertising interval set to $T_{GAP}$(dir_conn_adv_int_max). The Host shall configure the Controller to send directed connectable advertising events.

Note: When a device in the directed connectable mode establishes a connection, the device will exit this mode and enter the non-connectable mode.

### 9.3.4 Undirected Connectable Mode

#### *9.3.4.1 Description*

A device in the undirected connectable mode shall accept a connection request from a device performing the auto connection establishment procedure or the general connection establishment procedure.

#### *9.3.4.2 Conditions*

While a device is in the Peripheral role the device shall support the undirected connectable mode. While a device is in the Broadcaster, Observer, or the Central role the device shall not support the undirected connectable mode.

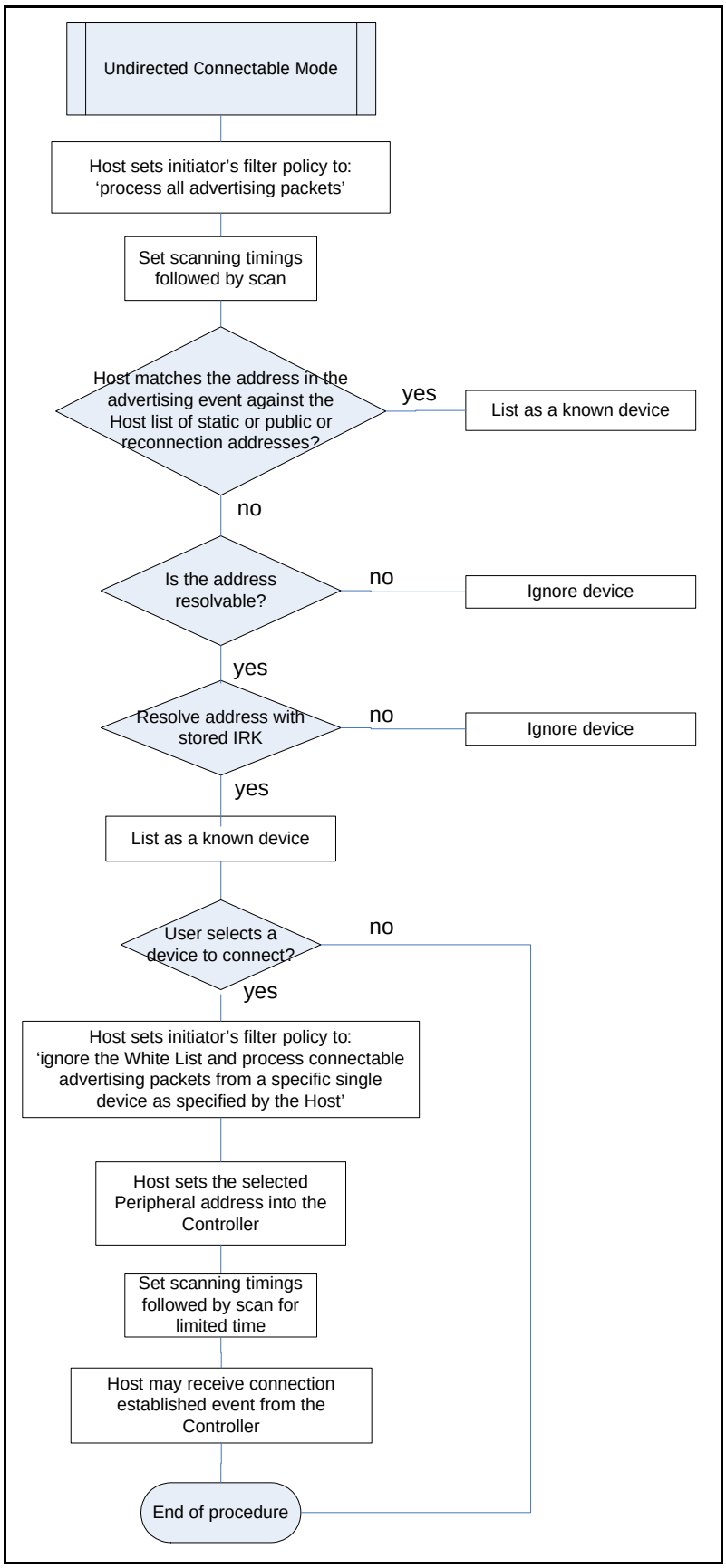



*Figure 9.4: Flow chart for a device entering the undirected connectable mode*

A Host entering the undirected connectable mode shall perform the following steps:

1. If the Peripheral Privacy Flag characteristic exists, the Host shall proceed to step 2, otherwise the Host shall proceed to step 4.
2. If the value in the Peripheral Privacy Flag characteristic is set to 1 (privacy enabled), the Host shall proceed to step 3, otherwise the Host shall proceed to step 4.
3. The Host shall generate a resolvable private address using the 'resolvable private address generation procedure' as defined in Section 10.8.2.2. The Host shall set a timer equal to or greater than $T_{GAP}$(private_addr_int). The Host shall configure the Controller to use the resolvable private address as the advertiser's device address in the connectable undirected advertising events. The Host should set the advertising filter policy to either 'process scan and connection requests only from devices in the White List' or 'process scan and connection requests from all devices'. The Host shall configure the Controller to send undirected connectable advertising events with the minimum advertising interval set to $T_{GAP}$(undir_conn_adv_int_min) and the maximum advertising interval set to $T_{GAP}$(undir_conn_adv_int_max). If privacy is disabled then the Host shall disable the $T_{GAP}$(private_addr_int) timer and proceed to step 4. When the $T_{GAP}$(private_addr_int) timer expires, the Host shall configure the Controller to stop advertising and repeat step 3.
4. The Host uses the static or the public address of the local device as the advertiser's device address. The Host should configure the Controller with the minimum advertising interval set to $T_{GAP}$(undir_conn_adv_int_min) and the maximum advertising interval set to $T_{GAP}$(undir_conn_adv_int_max). The Host shall configure the Controller to send undirected connectable advertising events.

Note: When a device in the undirected connectable mode establishes a connection, the device will exit this mode and enter the non-connectable mode.

### 9.3.5 Auto Connection Establishment Procedure

#### *9.3.5.1 Description*

The auto connection establishment procedure allows the Host to configure the Controller to autonomously establish a connection with one or more devices in the directed connectable mode or the undirected connectable mode. White Lists in the Controller are used to store device addresses. This procedure uses the initiator White List in the Controller. The Controller autonomously establishes a connection with a device with the device address that matches the address stored in the While List.

### *9.3.5.2 Conditions*

While a device is in the Central role the device may support the auto connection establishment procedure. While a device is in the Broadcaster, Observer, or Peripheral role the device shall not support the auto connection establishment procedure.

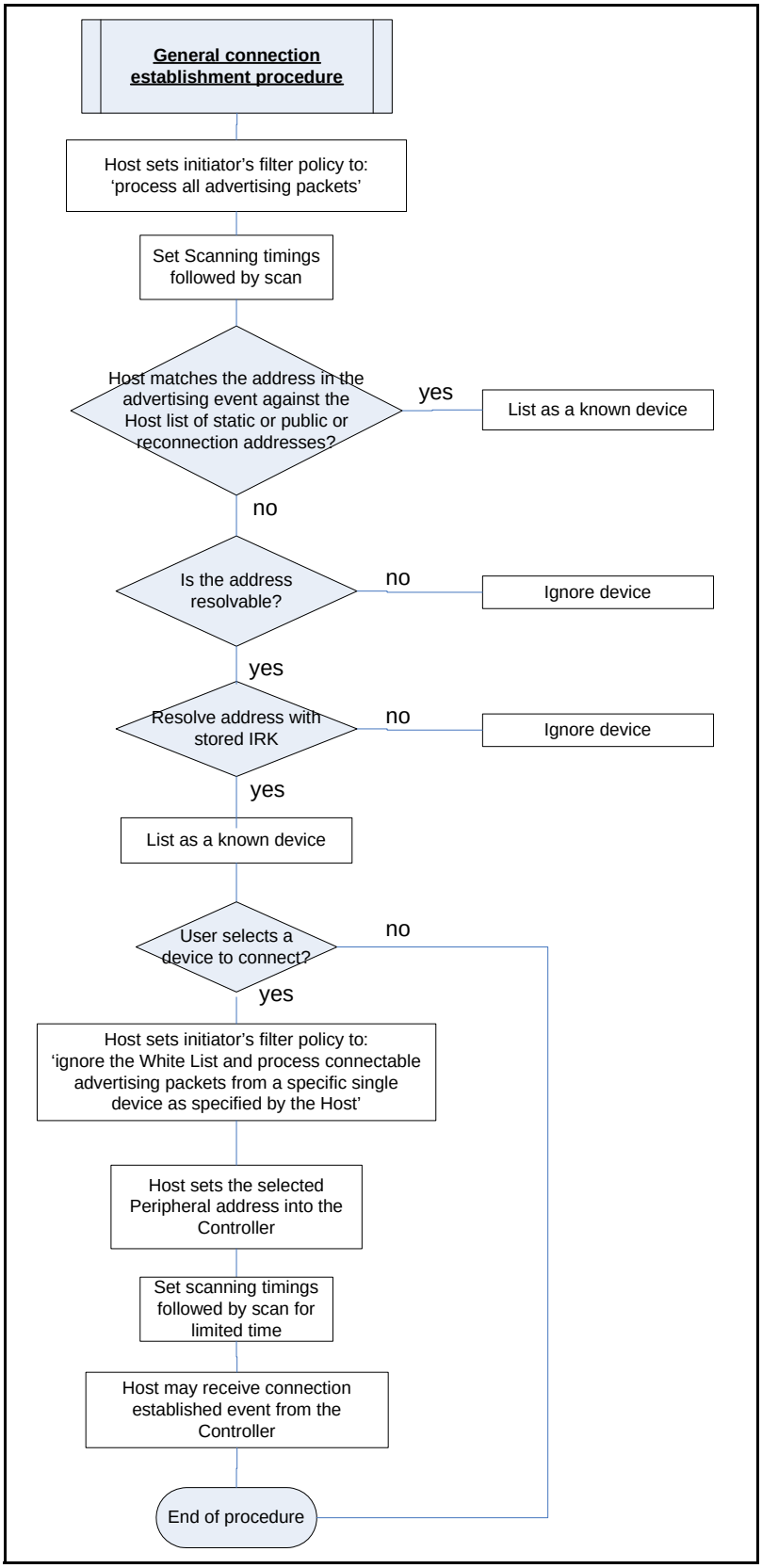



*Figure 9.5: Flow chart for a device performing the auto connection establishment procedure*

When a Host performs the auto connection establishment procedure, the Host configures the Controller as follows:

1. The Host shall write the list of device addresses that are to be auto connected to into the White List.
2. The Host shall set the initiator filter policy to 'process connectable advertising packets from all devices in the White List'.
3. The Host should set the minimum connection interval to be $T_{GAP}$(conn_est_int_min).
4. The Host should set the maximum connection interval to be $T_{GAP}$(conn_est_int_max)
5. The Host should set the slave latency to be $T_{GAP}$(conn_est_latency)
6. The Host should set the scan interval of $T_{GAP}$(auto_conn_est_scan_int) and the scan window of $T_{GAP}$(auto_conn_est_scan_wind).
7. The Host shall initiate a connection. For a privacy enabled device as defined in Section 10.7, the Host shall use a non-resolvable private address.

This procedure is terminated when a connection is established or when the Host terminates the procedure.

### 9.3.6 General Connection Establishment Procedure

#### *9.3.6.1 Description*

The general connection establishment procedure allows the Host to establish a connection with a set of known peer devices in the directed connectable mode or the undirected connectable mode.

#### *9.3.6.2 Conditions*

While a device is in the Central role the device shall support the general connection establishment procedure if the device supports the privacy feature, otherwise the device in the Central role may support the general connection establishment procedure. While a device is in the Broadcaster, Observer, or Peripheral role the device shall not support the general connection establishment procedure.

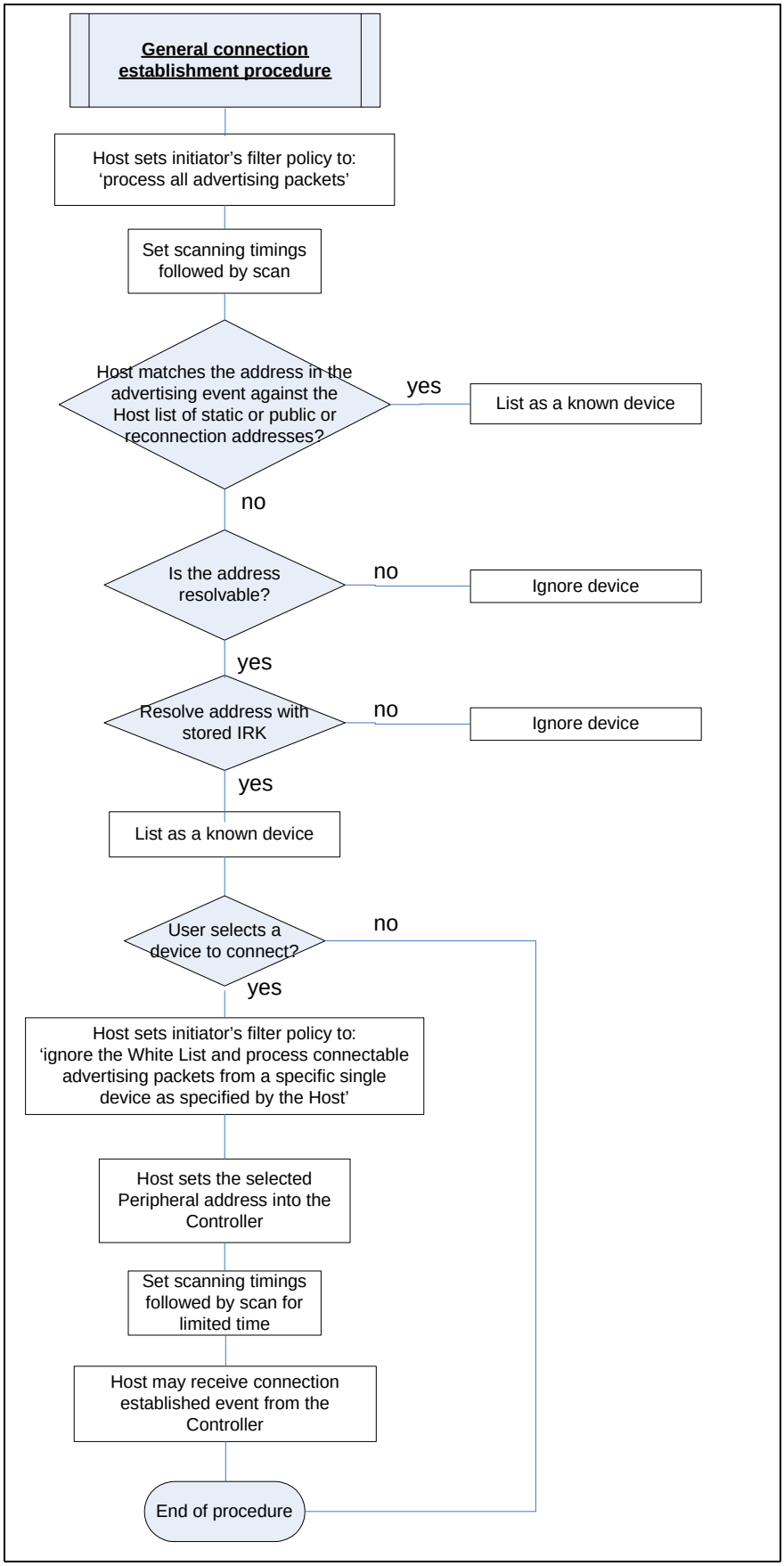



*Figure 9.6: Flow chart for a device performing the general connection establishment procedure*

When a Host performs the general connection establishment procedure, the Host configures the Controller as follows:

1. The Host shall set the scanner filter policy to 'process all advertising packets'.
2. The Host should set the scan interval to $T_{GAP}$(gen_conn_est_scan_int).
3. The Host should set the scan window to $T_{GAP}$(gen_conn_est_scan_wind).
4. The Host shall start scanning. For a privacy enabled device as defined in Section 10.7, the Host shall use a non-resolvable private address if the Host is in active scanning.

If the privacy feature is supported by the local device, and the Host discovers a device with a resolvable private address, the Host may resolve the resolvable private address by performing the 'resolvable private address resolution procedure' as defined in Section 10.8.2.3. If the resolution is successful, the Host considers the discovered device as a known device, otherwise the discovered device is ignored.

When the Host discovers a device to which the Host may attempt to connect, the Host shall stop the scanning, and initiate a connection using the direct connection establishment procedure.

When the Host of the Central that supports privacy establishes a connection with a Peripheral that has the 'Peripheral Privacy Flag' set to 1 (privacy enabled), the Host of the Central should generate a new reconnection address and update the reconnection address characteristic value in the Peripheral before any application data is sent by the Host of the Central. The Peripheral shall use the old reconnection address if the reconnection address has not been updated.

If the Central does not support privacy, when the Host of the Central establishes a connection with a Peripheral that has the 'Peripheral Privacy Flag' set to 1 (privacy enabled), the Host of the Central may set the Peripheral Privacy Flag to 0 (privacy disabled). If the Host of the Central is unable to disable the Peripheral Privacy Flag, the Central should not bond with the Peripheral.

This procedure is terminated when a connection is established or when the Host terminates the procedure.

### 9.3.7 Selective Connection Establishment Procedure

#### *9.3.7.1 Description*

The selective connection establishment procedure allows the Host to establish a connection with the Host selected connection configuration parameters with a set of devices in the White List.

#### *9.3.7.2 Conditions*

While a device is in the Central role the device may support the selective connection establishment procedure. While a device is in the Broadcaster, Observer, or the Peripheral role the device shall not support the selective connection establishment procedure.

When a Host performs the selective connection establishment procedure, the Host configures the Controller as follows:

1. The Host shall write the list of device addresses that are to be selectively connected to into the White List.
2. The Host shall set the scanner filter policy to ‘process advertising packets only from devices in the White List’.
3. The Host should set the scan interval to $T_{GAP}$(sel_conn_est_scan_int).
4. The Host should set the scan window to $T_{GAP}$(sel_conn_est_scan_wind).
5. The Host shall start active scanning or passive scanning.

When the Host discovers one of the peer devices it is connecting to, the Host shall stop scanning, and initiate a connection using the direct connection establishment procedure with the connection configuration parameters for that peer device.

This procedure is terminated when a connection is established or when the Host terminates the procedure.

### 9.3.8 Direct Connection Establishment Procedure

#### *9.3.8.1 Description*

The direct connection establishment procedure allows the Host to establish a connection with the Host selected connection configuration parameters with a single peer device.

#### *9.3.8.2 Conditions*

While a device is in the Central role the device shall support the direct connection establishment procedure. While a device is in the Broadcaster, Observer, or the Peripheral role the device shall not support the direct connection establishment procedure.

When a Host performs the direct connection establishment procedure, the Host configures the Controller as follows:

1. The Host shall set the initiator filter policy to ‘ignore the White List and process connectable advertising packets from a specific single device specified by the Host’.

2. The Host shall set the peer address to the device address of the specific single device.
3. The Host shall set the connection configuration parameters for the peer device.

The Host shall initiate a connection. For a privacy enabled device as defined in Section 10.7, the Host shall use a non-resolvable private address.

This procedure is terminated when a connection is established or when the Host terminates the procedure.

### 9.3.9 Connection Parameter Update Procedure

#### *9.3.9.1 Description*

The connection parameter update procedure allows a Peripheral or Central to update the Link Layer connection parameters of an established connection.

#### *9.3.9.2 Conditions*

While a device is in the Central role the device shall support the connection parameter update procedure. While a device is in the Peripheral role the device may support the connection parameter update procedure. While a device is in the Broadcaster or the Observer role the device shall not support the connection parameter update procedure.

A Central initiating the connection parameter update procedure shall use the Link Layer Connection Update procedure defined in [Vol. 6], Part B Section 5.1.1 with the required connection parameters.

A Peripheral initiating the connection parameter update procedure shall use the L2CAP Connection Parameter Update Request command defined in [Vol. 1], Part A Section 4.2 with the required connection parameters. The Peripheral shall not send an L2CAP Connection Parameter Update Request command within $T_{GAP}$(conn_param_timeout) of an L2CAP Connection Parameter Update Response being received. If the updated connection parameters are unacceptable to the Peripheral then it may disconnect the connection with the error code «Unacceptable Connection Interval». Peripherals should be tolerant of connection intervals given to it from the Central.

### 9.3.10 Terminate Connection Procedure

#### *9.3.10.1 Description*

The terminate connection procedure allows a Host to terminate the connection with a peer device.

#### *9.3.10.2 Conditions*

Centrals and Peripherals shall support the terminate connection procedure. Broadcasters and Observers shall not support the terminate connection procedure.

The Host initiating the terminate connection procedure shall use the Link Layer Termination Procedure defined in [Vol. 6], Part B Section 5.1.6.

## 9.4 BONDING MODES AND PROCEDURES

Bonding allows two connected devices to exchange and store security and identity information to create a trusted relationship. The security and identity information as defined in [Vol. 3], Part H Section 2.4.1 is also known as the bonding information. When the devices store the bonding information, it is known as the phrases ‘devices have bonded’ or ‘a bond is created’.

There are two modes for bonding, non-bondable mode and bondable mode. Bonding may only occur between two devices in bondable mode. The requirements for a device to support the bonding modes and procedure are defined in Table 9.4.

### 9.4.1 Requirements

| Bonding | Ref. | Peripheral | Central |
|---|---|---|---|
| Non-Bondable mode | 9.4.2 | M | M |
| Bondable mode | 9.4.3 | O | O |
| Bonding procedure | 9.4.4 | O | O |

*Table 9.4: Bonding Compliance Requirements*

### 9.4.2 Non-Bondable Mode

#### *9.4.2.1 Description*

A device in the non-bondable mode does not allow a bond to be created with a peer device.

#### *9.4.2.2 Conditions*

While a device is in the Peripheral or the Central role the device shall support the non-bondable mode. If a device does not support pairing as defined in the Security Manager section then it is considered to be in non-bondable mode.

If Security Manager pairing is supported, the Host shall set the Bonding_Flags to ‘No Bonding’ as defined in [Vol. 3], Part H Section 3.5.1 and bonding information shall not be exchanged or stored.

### 9.4.3 Bondable Mode

#### *9.4.3.1 Description*

A device in the bondable mode allows a bond to be created with a peer device in the bondable mode.

#### *9.4.3.2 Conditions*

The Host shall set the Bonding_Flags to ‘General Bonding’ during the pairing procedure.

### 9.4.4 Bonding Procedure

#### *9.4.4.1 Description*

The bonding procedure may be performed when a non-bonded device tries to access a service that requires bonding. The bonding procedure may be performed for the purpose of creating a bond between two devices.

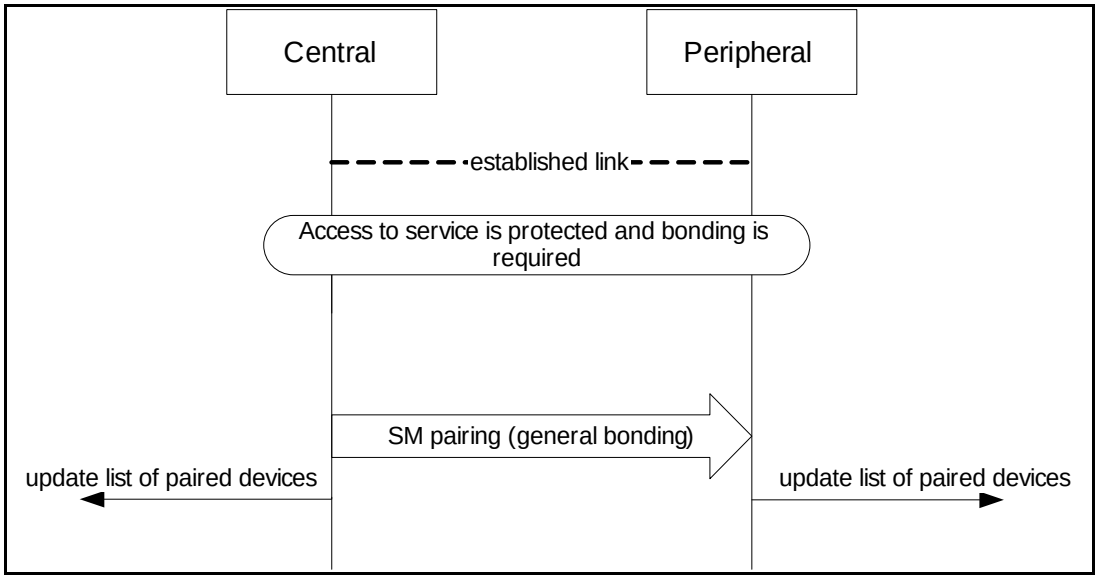



*Figure 9.7: The Bonding procedure*

#### *9.4.4.2 Conditions*

While a device is in the Peripheral or the Central role the device may support the Bonding procedure. While a device is in the Broadcaster or the Observer role the device shall not support the bonding procedure.

The Host of the Central initiates the pairing process as defined in [Vol. 3], Part C Section 2.1 with the Bonding_Flags set to Bonding as defined in [Vol. 3], Part H Section 3.5.1. If the peer device is in the bondable mode, the devices shall exchange and store the bonding information in the security database.

If a device has privacy enabled as defined in Table 10.7, the Host should send it's IRK to the peer device and request the IRK of the peer device during the pairing procedure. The Host can abort the pairing procedure if the authentication requirements are not sufficient to distribute the IRK. If the pairing procedure fails due to authentication requirements and IRK distribution was requested, the pairing procedure should be retried without requesting IRK distribution.

# 10 LE SECURITY ASPECTS

This section defines the modes and procedures that relate to the security of a connection. The following modes and procedures are defined:

- LE security mode 1
- LE security mode 2
- Authentication procedure
- Authorization procedure
- Connection data signing procedure
- Authenticate signed data procedure

Requirements for a device to support the LE security modes and procedures is shown in Table 10.1.

## 10.1 REQUIREMENTS

| Security Modes and Procedures | Ref. | Broadcaster | Observer | Peripheral | Central |
|---|---|---|---|---|---|
| LE Security mode 1 | 10.2.1 | E | E | O | O |
| LE Security mode 2 | 10.2.2 | E | E | O | O |
| Authentication procedure | 10.3 | E | E | O | O |
| Authorization procedure | 10.5 | E | E | O | O |
| Connection data signing procedure | 10.4.1 | E | E | O | O |
| Authenticate signed data procedure | 10.4.2 | E | E | O | O |

*Table 10.1: Requirements related to security modes and procedures*

## 10.2 LE SECURITY MODES

The security requirements of a device, a service or a service request are expressed in terms of a security mode and security level. Each service or service request may have its own security requirement. The device may also have a security requirement. A physical connection between two devices shall operate in only one security mode.

There are two LE security modes, LE security mode 1 and LE security mode 2.

### 10.2.1 LE Security Mode 1

LE security mode 1 has three security levels:

1. No security (No authentication and no encryption)
2. Unauthenticated pairing with encryption
3. Authenticated pairing with encryption

A connection operating in LE security mode 1 level 2 shall also satisfy the security requirements for LE security mode 1 level 1.

A connection operating in LE security mode 1 level 3 shall also satisfy the security requirements for LE security mode 1 level 2 or LE security mode 1 level 1.

A connection operating in LE security mode 1 level 3 shall also satisfy the security requirements for LE security mode 2.

### 10.2.2 LE Security Mode 2

LE security mode 2 has two security levels:

1. Unauthenticated pairing with data signing
2. Authenticated pairing with data signing

LE security mode 2 shall only be used for connection based data signing.

Data signing as defined in Section 10.4 shall not be used when a connection is operating in LE security mode 1 level 2 or LE security mode 1 level 3.

### 10.2.3 Mixed Security Modes Requirements

If there are requirements for both LE security mode 1 and LE security mode 2 level 2 for a given physical link then LE security mode 1 level 3 shall be used.

If there are requirements for both LE security mode 1 level 3 and LE security mode 2 for a given physical link then LE security mode 1 level 3 shall be used.

If there are requirements for both LE security mode 1 level 2 and LE security mode 2 level 1 for a given physical link then LE security mode 1 level 2 shall be used.

## 10.3 AUTHENTICATION PROCEDURE

The authentication procedure describes how the required security is established when a device initiates a service request to a remote device and when a device receives a service request from a remote device. The authentication procedure covers both LE security mode 1 and LE security mode 2. The authentication procedure shall only be initiated after a connection has been established.

There are two types of pairing: authenticated pairing or unauthenticated pairing. Authenticated pairing involves performing the pairing procedure defined in [Vol. 3], Part H Section 2.1 with the authentication set to ‘MITM protection’. Unauthenticated pairing involves performing the pairing procedure with authentication set to ‘No MITM protection’.

### 10.3.1 Receiving a Service Request

A service request is a request to perform a GATT procedure. A service request may also be a request to perform a procedure using other higher layer protocols on top of L2CAP.

When a device receives a service request from a remote device it shall behave as follows:

- The Server’s security database specifies the security settings required to accept a service request from a remote device. If no security is required, the service request shall be accepted; otherwise pairing is required before the service request can be accepted.
- If pairing has not occurred or an authenticated pairing is required but only an unauthenticated pairing has occurred, the service request shall be rejected with the error code “Insufficient Authentication“.
- If the required pairing has occurred and encryption is required (LE security mode 1) but encryption is not enabled, the service request shall be rejected with the error code “Insufficient Encryption”. If the encryption is enabled with insufficient key size then the service request shall be rejected with the error code “Insufficient Encryption Key Size.”
- If authenticated pairing has occurred and encryption is not required (LE security mode 2) then Data Signing shall be used if CSRK has been exchanged, else the Host shall re-pair and exchange CSRK before proceeding with Data Signing.

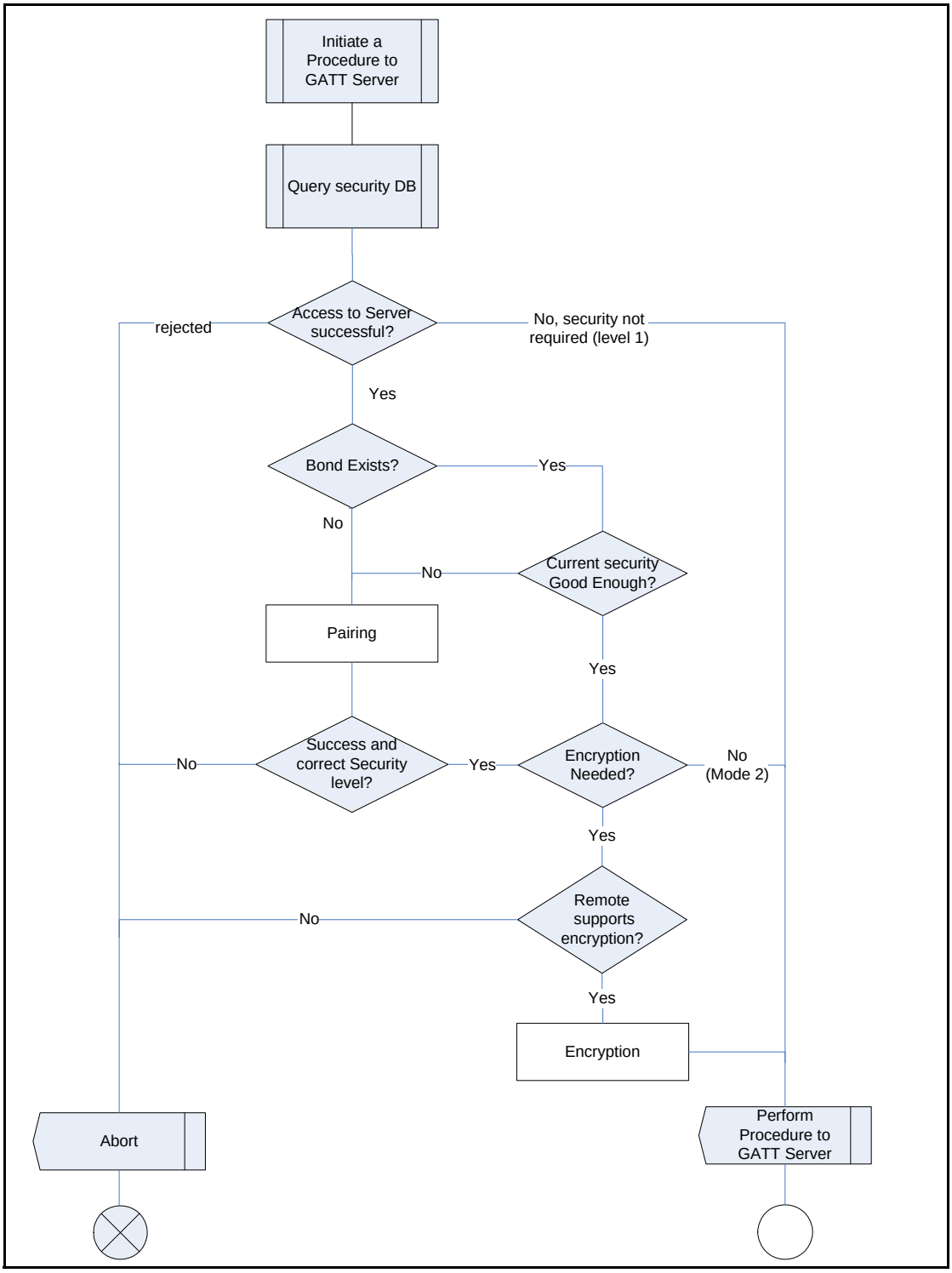



*Figure 10.1: Flow chart for a server handling the service request*

### 10.3.2 Initiating a Service Request

When a device initiates a service request to a remote device it shall behave as follows:

- The Client's security database specifies the security required to initiate a service request. If no security is required then the service request shall proceed otherwise pairing is required. If security is required but pairing has not occurred in a previous bonding procedure then pairing shall occur during the current connection.
- If pairing has not occurred or an authenticated pairing is required but only an unauthenticated pairing has occurred, the pairing procedure shall be executed with the required authentication settings. If the pairing procedure fails then the service request shall be aborted.
- If authenticated pairing has occurred but the encryption key size is insufficient the pairing procedure shall be executed with the required encryption key size. If the pairing procedure fails then the service request shall be aborted.
- If the required pairing has occurred and encryption is required (LE security mode 1) then encryption shall be enabled before the service request proceeds. If encryption is required but encryption is not enabled and the remote device supports encryption then encryption shall be enabled using the encryption procedure as defined in Section 10.6. Once encryption is enabled the service request shall proceed. If encryption is required but not supported by the remote device then the service request shall be aborted.
- If encryption is not required and authenticated pairing has occurred (LE security mode 2) then the data signing procedure shall be used when making the service request.

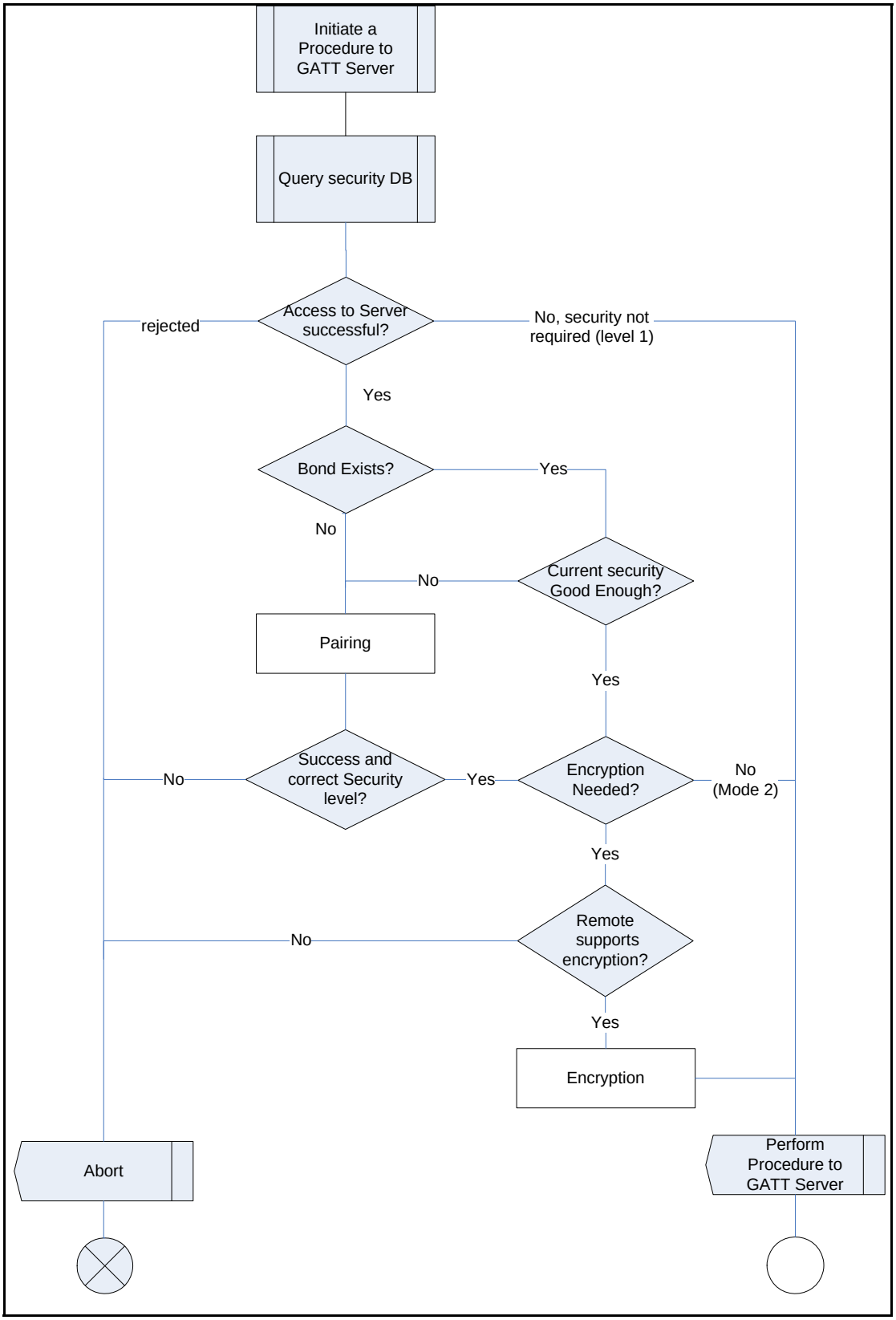



*Figure 10.2: Flow chart for a client issuing a service request*

## 10.4 DATA SIGNING

The data signing is used for transferring authenticated data between two devices in an unencrypted connection. The data signing method is used by services that require fast connection set up and fast data transfer.

If a service request specifies LE security mode 2, the connection data signing procedure shall be used.

### 10.4.1 Connection Data Signing Procedure

A device shall generate a new Connection Signature Resolving Key CSRK for each peer device to which it sends signed data in a connection. CSRK is defined in [Vol. 3], Part H Section 2.4.2.2.

The data shall be formatted using the Signing Algorithm as defined in [Vol. 3], Part H Section 2.4.5 where m is the Data PDU to be signed, k is the CSRK and the SignCounter is the counter value. A Signature is composed of the counter value and the Message Authentication Code (MAC) generated by the Signing Algorithm. The counter value shall be incremented by one for each new Data PDU sent.

The format of signed data is shown in Figure 10.3.

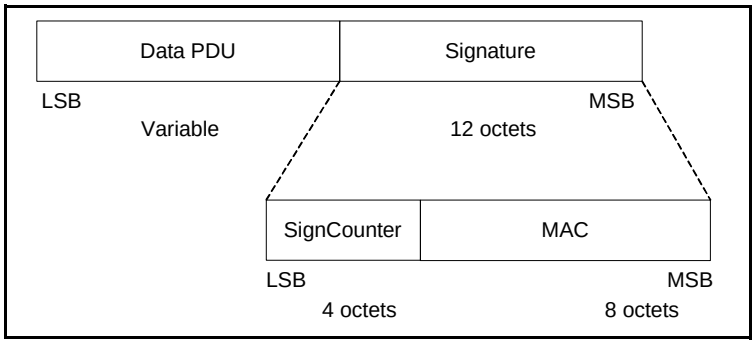



*Figure 10.3: Generic format of signed data*

### 10.4.2 Authenticate Signed Data Procedure

A device connected to a peer device sending signed data can authenticate the signed data by using the stored CSRK.

The signed data shall be authenticated by performing the Signing Algorithm where m is the Data PDU to be authenticated, k is the stored CSRK and the SignCounter is the received counter value. If the MAC computed by the Signing Algorithm does not match the received MAC, the verification fails and the Host shall ignore the received Data PDU

The receiving device shall protect against a replay attack by comparing the received SignCounter with previously received SignCounter from the same

peer device. If the SignCounter was previously used then the receiving device shall ignore the Data PDU and respond with error code “Signature Authentication Failed”, if permitted. The receiving device should protect from repeated failures by implementing a timer similar to the pairing timer as defined in [Vol. 3], Part H Section 2.3.6.

## 10.5 AUTHORIZATION PROCEDURE

A service may require authorization before allowing access. Authorization is a confirmation by the user to continue with the procedure. Authentication does not necessarily provide authorization. Authorization may be granted by user confirmation after successful authentication.

## 10.6 ENCRYPTION PROCEDURE

A Central may encrypt a connection using the Encryption Session Setup as defined in [Vol. 3], Part H Section 2.4.4 to provide integrity and confidentiality.

A Peripheral may encrypt a connection using the Slave Initiated Security Request as defined in [Vol. 3], Part H Section 2.4.6 to provide integrity and confidentiality.

## 10.7 PRIVACY FEATURE

The privacy feature provides a level of privacy which makes it more difficult for an attacker to track a device over a period of time. The requirements for a device to support the privacy feature are defined in Table 10.2.

| Privacy Requirements | Ref. | Broadcaster | Observer | Peripheral | Central |
|---|---|---|---|---|---|
| Privacy feature | 10.7 | O | O | O | O |
| Non-resolvable private address generation procedure | 10.8.2.1 | E | C1 | C2 | C1 |
| Resolvable private address generation procedure | 10.8.2.2 | C3 | E | C3 | E |
| Resolvable private address resolution procedure | 10.8.2.3 | E | C2 | E | C3 |
| C1: Mandatory if privacy feature and Active scanning are supported, else excluded<br>C2: Optional if privacy is supported, else excluded<br>C3: Mandatory if privacy feature is supported, else excluded | | | | | |

*Table 10.2: Requirements related to privacy feature*

### 10.7.1 Privacy Feature in a Peripheral

Support of the privacy feature in a Peripheral depends on the presence and the value of two characteristics: Peripheral Privacy Flag characteristic as defined in Section 12.3 and the Reconnection Address characteristic as defined in Section 12.4. When the Peripheral Privacy Flag characteristic is set to 1, privacy is enabled (also known as privacy enabled Peripheral). Table 10.3 shows privacy support in the Peripheral. A privacy enabled Peripheral that does not expose the Reconnection Address characteristic may use a resolvable private address as the advertiser's device address when sending advertising events, depending on the mode that the Peripheral is operating in. A privacy enabled Peripheral that does expose the Reconnection Address characteristic may use the reconnection address as the advertiser's device address only if it has been set by the Central when sending advertising events depending on the mode that the Peripheral is operating in. When the Peripheral Privacy Flag characteristic is set to 0, privacy is disabled. When privacy is disabled the peripheral shall use a static or public address. When privacy is disabled, the permission of the Reconnection Address characteristic value shall be read-only, and shall not be considered as set by the Central.

| Peripheral Privacy Flag characteristic present and set to 1 (privacy enabled)? | Reconnection Address characteristic present? | Privacy support in Peripheral |
|---|---|---|
| No | No | No privacy |
| Yes | No | Privacy in undirected connectable mode only |
| No | Yes | No privacy |
| Yes | Yes | Privacy in directed connectable mode and in undirected connectable mode |

*Table 10.3: privacy feature in a Peripheral*

Note: A privacy enabled Peripheral may write one or more Central's static or public address in the White List of the Controller. A privacy enabled Peripheral may also write one or more reconnection addresses in the White List of the Controller. Static, public and reconnection addresses stored in the White List allow the Controller of the Peripheral to autonomously establish a connection with the Central that initiates a connection with the address that matches the address in the White List.

A Peripheral that supports the Peripheral Privacy Flag characteristic may allow a bonded Central to enable or disable the privacy feature. If the number of bonds on the Peripheral is less than or equal to one, the Peripheral Privacy Flag characteristic value may be writable. If a Peripheral has more than one bond, the permission of the Peripheral Privacy Flag characteristic value shall be read-only.

### 10.7.2 Privacy Feature in a Central

A device in the Central may support the privacy feature by use of the non-resolvable address or the reconnection address as the device address.

If the privacy feature is supported in a Central (also known as privacy enabled Central), the Central shall use a non-resolvable private address during active scanning.

When a Central connects to a privacy enabled Peripheral with the reconnection address characteristic, the Central shall write a new reconnection address in the reconnection address characteristic in the Peripheral. See general connection establishment procedure in Section 9.3.6. The privacy enabled Central may use the reconnection address when initiating a connection to a privacy enabled Peripheral.

When a Central connects with a Peripheral that supports the Peripheral Privacy Flag characteristic the Central may be allowed to enable or disable the privacy feature in the Peripheral.

Note: A privacy enabled Central can write the reconnection address in the White List.

## 10.8 RANDOM DEVICE ADDRESS

Random device addresses are defined in the Link Layer section in [Vol. 6], Part B Section 1.3.

For the purposes of this profile, the random device address may be of either of the following two sub-types:

- Static address
- Private address.

The term random device address refers to both static and private address types.

The transmission of a random device address is optional. A device shall accept the reception of a random device address from a remote device.

The private address may be of either of the following two sub-types:

- Non-resolvable private address
- Resolvable private address

### 10.8.1 Static Address

A static address is a 48-bit randomly generated address and shall meet the following requirements:

- The two most significant bits of the static address shall be equal to ‘1’
- All bits of the random part of the static address shall not be equal to ‘1’
- All bits of the random part of the static address shall not be equal to ‘0’

The format of a static address is shown in Figure 10.4.

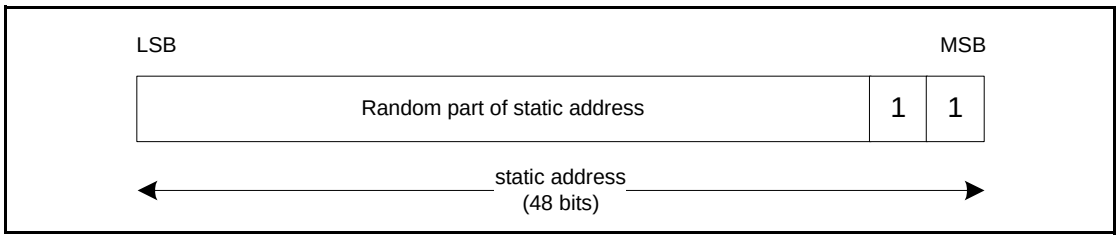



*Figure 10.4: Format of static address*

A device may choose to initialize its static address to a new value after each power cycle. A device shall not change its static address value once initialized until the device is power cycled.

Note: If the static address of a device is changed then the address stored in peer devices will not be valid and the ability to reconnect using the old address will be lost.

### 10.8.2 Private address

The private address may be of either of the following two sub-types:

- Non-resolvable private address
- Resolvable private address

The generation of a reconnection address shall use the non-resolvable private address generation procedure.

#### *10.8.2.1 Non-Resolvable Private Address Generation Procedure*

To generate a non-resolvable address or a reconnection address, the Host shall generate a 48-bit address with the following requirements:

- The two most significant bits of the address shall be equal to ‘0’
- All bits of the random part of the address shall not be equal to ‘0’
- All bits of the random part of the address shall not be equal to ‘1’
- The address shall not be equal to the static address
- The address shall not be equal to the public address

The format of a non-resolvable private address is shown in Table 10.5.

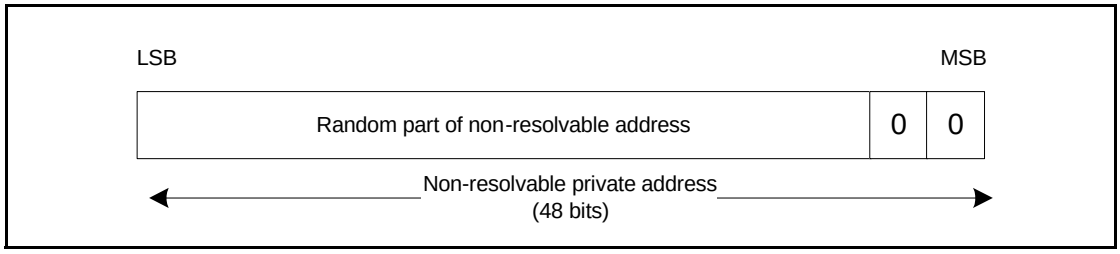



*Figure 10.5: Format of non-resolvable private address*

### *10.8.2.2 Resolvable Private Address Generation Procedure*

To generate a resolvable private address the Host must have its Identity Resolving Key (IRK). The resolvable private address shall be generated with the IRK and a randomly generated 24-bit random number. The random number is known as *prand* and shall meet the following requirements:

- The two most significant bits of *prand* shall be equal to '0' and '1' as shown in Figure 10.6
- All bits of the random field of *prand* shall not be equal to '0'
- All bits of the random field of *prand* shall not be equal to '1'

The format of the resolvable private address is shown in Figure 10.6.

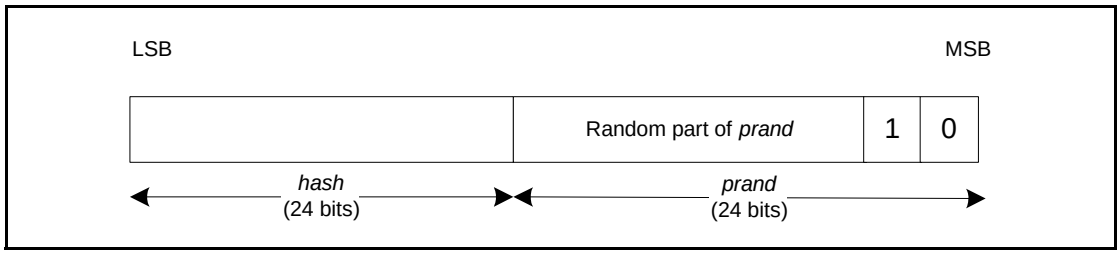



*Figure 10.6: Format of resolvable private address*

The *hash* is generated using the random address function *ah* defined in [Vol. 3], Part H Section 2.2.2 with the input parameter *k* set to the device's IRK and the input parameter *r* set to *prand*.

*hash* = *ah*(IRK, *prand*)

The *prand* and *hash* are concatenated to generate the random address (*randomAddress*) in the following manner:

*randomAddress* = *hash* || *prand*

The least significant octet of *hash* becomes the least significant octet of *randomAddress* and the most significant octet of *prand* becomes the most significant octet of *randomAddress*.

### *10.8.2.3 Resolvable Private Address Resolution Procedure*

The Host can resolve a resolvable private address for all peer devices where the Host has the peer device's IRK using this procedure. If a resolvable private address is resolved, the Host can associate this address with the peer device.

The resolvable private address (*rpAddress*) is divided into a 24-bit random part (*prand*) and a 24-bit hash part (*hash*). The least significant octet of the *rpAddress* becomes the least significant octet of *hash* and the most significant octet of *rpAddress* becomes the most significant octet of *prand*.

A *localHash* value is then generated using the random address hash function *ah* defined in [Vol. 3], Part H Section 2.2.2 with the input parameter *k* set to IRK of the known device and the input parameter *r* set to the *prand* value extracted from the *rpAddress*.

*localHash* = *ah*(IRK, *prand*)

The *localHash* value is then compared with the *hash* value extracted from *rpAddress*. If the localHash value matches the extracted *hash* value then the identity of the peer device has been resolved.

If a Host has more than one stored IRK, the Host repeats the above procedure for each stored IRK to determine if the received resolvable private address is associated with a stored IRK, until either address resolution is successful for one of the IRKs or all have been tried.

# 11 ADVERTISING AND SCAN RESPONSE DATA FORMAT

The format of Advertising data and Scan Response data is shown in Figure 11.1. The data consists of a significant part and a non-significant part. The significant part contains a sequence of AD structures. Each AD structure shall have a Length field of one octet, which contains the Length value, and a Data field of Length octets. The first octet of the Data field contains the AD type field. The content of the remaining Length – 1 octet in the Data field depends on the value of the AD type field and is called the AD data. The non-significant part extends the Advertising and Scan Response data to 31 octets and shall contain all-zero octets.

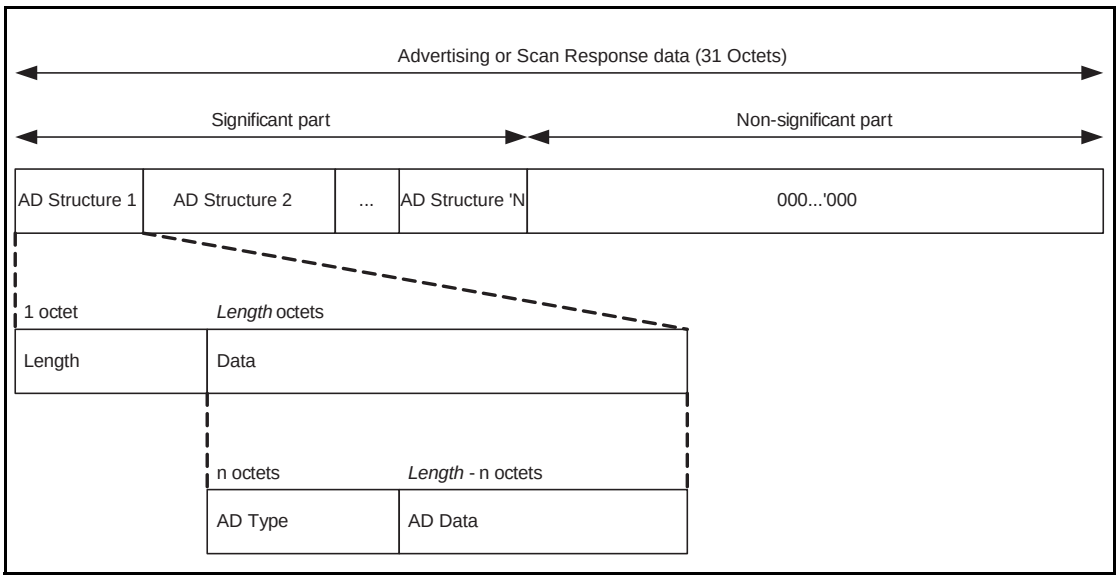



*Figure 11.1: Advertising and Scan Response data format*

If the Length field is set to zero, then the Data field has zero octets. This shall only occur to allow an early termination of the Advertising or Scan Response data.

Only the significant part of the Advertising or Scan Response data needs to be sent over the air.

The Advertising and Scan Response data is sent in advertising events. The Advertising Data is placed in the AdvData field of ADV_IND, ADV_NONCONN_IND, and ADV_SCAN_IND packets. The Scan Response data is sent in the ScanRspData field of SCAN_RSP packets.

## 11.1 AD TYPE DEFINITIONS

All AD types are listed in the Bluetooth Assigned Numbers document.

All numerical multi-byte entities and values shall use little-endian byte order.

### 11.1.1 Service UUIDs

A device may list the service UUIDs that it implements in its advertising data. Service UUIDs may be either 16-bit UUIDs or 128-bit UUIDs. The format of Service UUIDs for each type value is defined in Table 18.2.

### 11.1.2 Local Name

The Local Name AD type contains the device name, either complete or shortened as defined in Section 3.2.2. The AD type indicates if the name is complete or shortened. If the name is shortened, the complete device name can be read by reading the device name characteristic as defined in Section 12. The Advertising and Scan Response data shall not contain more than one instance of the Local Name AD type. A shortened name shall only contain contiguous characters from the beginning of the full name. For example, if the device name is ‘BT_Device_Name’ then the shortened name over BR/EDR could be ‘BT_Device’ while the shortened name on LE could be ‘BT_Dev’. The format of the data is defined in Table 18.3.

### 11.1.3 Flags

The Flags AD type contains several flag bits interpreted as boolean values.

Flags used over the LE physical channel are:

- Limited Discoverable Mode
- General Discoverable Mode
- BR/EDR Not Supported
- Simultaneous LE and BR/EDR to Same Device Capable (Controller)
- Simultaneous LE and BR/EDR to Same Device Capable (Host)

The Flags AD type shall not be included in the scan response data. The advertising data shall not contain more than one instance of the Flags AD type. The Flags AD type shall be included in the advertising data if any of the bits are non-zero. The Flags AD type may be omitted from the advertising data if all of the bits are zero. The format of Flags AD type is defined in Table 18.1.

### 11.1.4 Manufacturer Specific Data

The Manufacturer Specific AD type is used for manufacturer specific data. See Section 18.11.

### 11.1.5 TX Power Level

The TX Power Level AD type indicates the transmitted power level of the advertising packet. The format of TX Power Level AD is defined in Table 18.4.

### 11.1.6 Security Manager Out of Band (OOB)

An out of band mechanism is used by the Security Manager to communicate discovery information as well as other information related to the pairing process. The format of Security Manager Out of Band AD is defined in Section 18.7. This shall only be used over an out-of-band mechanism.

### 11.1.7 Security Manager TK Value

The Security Manager TK Value AD type is defined in Section 18.6. This shall only be used over an out-of-band mechanism.

### 11.1.8 Slave Connection Interval Range

The Slave Connection Interval Range AD type contains the Peripheral's preferred connection interval range, for all logical connections.

Note: The minimum value depends on the battery considerations of the Peripheral and the maximum connection interval depends on the buffers available on the Peripheral.

The Central should use the information from the Peripheral's Slave Connection Interval Range AD type when establishing a connection.

The Slave Connection Interval Range AD type is defined in Section 18.8**.**

### 11.1.9 Service Solicitation

One of the Service Solicitation AD types may be sent to invite other devices that expose one or more of the services specified in the Service Solicitation data to connect. The device should be in the undirected connectable mode and in one of the discoverable modes. This enables a Central providing one or more of these services to connect to this Peripheral, so that the Peripheral can use the services on the Central.

The Service Solicitation AD types are defined in Section 18.9.

### 11.1.10 Service Data

The Service Data AD type consists of a service UUID with the data associated with that service.

The Service Data AD type is defined in Section 18.10.

## 11.2 EXAMPLE ADVERTISING DATA

This is an example of advertising data with AD types:

| Value | Notes |
|---|---|
| 0x02 | Length of this Data |
| 0x01 | AD type = Flags |
| 0x01 | LE Limited Discoverable Flag set |
| 0x0A | Length of this Data |
| 0x09 | AD type = Complete local name |
| 0x50 | ‘P’ |
| 0x65 | ‘e’ |
| 0x64 | ‘d’ |
| 0x6F | ‘o’ |
| 0x6D | ‘m’ |
| 0x65 | ‘e’ |
| 0x74 | ‘t’ |
| 0x65 | ‘e’ |
| 0x72 | ‘r’ |

# 12 GAP CHARACTERISTICS FOR LOW ENERGY

The service UUID shall be the UUID for «Generic Access Profile.»

The characteristics requirements for a device in each of the GAP roles are shown in Table 12.1. A device shall have only one instance of the GAP service. The GAP service is a GATT based service with the service UUID as defined in the Bluetooth Assigned Numbers document.

| Characteristics | Ref. | Broadcaster | Observer | Peripheral | Central |
|---|---|---|---|---|---|
| Device Name | 12.1 | E | E | M | M |
| Appearance | 12.2 | E | E | M | M |
| Peripheral Privacy Flag | 12.3 | E | E | C1 | E |
| Reconnection Address | 12.4 | E | E | C2 | E |
| Peripheral Preferred Connection Parameters | 12.5 | E | E | O | E |
| C1: Mandatory if privacy feature is supported, else excluded<br>C2: Optional if privacy feature and Peripheral privacy Flag characteristic are supported, else excluded | | | | | |

*Table 12.1: Requirements related to characteristics*

A device that supports multiple GAP roles contains all the characteristics to meet the requirements for the supported roles. The device must continue to expose the characteristics when the device is operating in the role in which the characteristics are not valid.

## 12.1 DEVICE NAME CHARACTERISTIC

The Device Name characteristic shall contain the name of the device as an UTF-8 string as defined in Section 3.2.2. The Device Name characteristic value shall be readable without authentication or authorization. The Device Name characteristic value may be writable.

| Attribute Handle | Attribute Type | Attribute Value |
|---|---|---|
| 0xMMMM | 0x2A00 – UUID for «Device Name» | Device Name |

*Table 12.2: Device Name characteristic*

The Device Name characteristic value shall be 0 to 248 octets in length. A device shall have only one instance of the Device Name characteristic.

## 12.2 APPEARANCE CHARACTERISTIC

The Appearance characteristic defines the representation of the external appearance of the device. This enables the discovering device to represent the device to the user using an icon, or a string, or similar. The Appearance characteristic value shall be readable without authentication or authorization. The Appearance characteristic value may be writable.

| Attribute Handle | Attribute Type | Attribute Value |
|---|---|---|
| 0xMMMM | 0x2A01 – UUID for «Appearance» | Appearance |

*Table 12.3: Appearance characteristic*

The Appearance characteristic value shall be the enumerated value as defined by Bluetooth Assigned Numbers document. The Appearance characteristic value shall be 2 octets in length. A device shall have only one instance of the Appearance characteristic.

## 12.3 PERIPHERAL PRIVACY FLAG CHARACTERISTIC

The Peripheral Privacy Flag characteristic defines whether privacy is currently in use within this device. See Table 10.7.2.

| Attribute Handle | Attribute Type | Attribute Value |
|---|---|---|
| 0xMMMM | 0x2A02 – UUID for «Peripheral Privacy Flag» | Peripheral Privacy Flag |

*Table 12.4: Peripheral Privacy Flag characteristic*

The Peripheral Privacy Flag characteristic value shall be 1 octet in length.

0 = privacy is disabled in this device

1 = privacy is enabled in this device

2 – 255 = Reserved

A device shall have only one instance of the Peripheral Privacy Flag characteristic.

## 12.4 RECONNECTION ADDRESS CHARACTERISTIC

The Reconnection Address characteristic defines the reconnection address. See Table 10.7.2. The reconnection address is a non-resolvable private address.

| Attribute Handle | Attribute Type | Attribute Value |
|---|---|---|
| 0xMMMM | 0x2A03 – UUID for «Reconnection Address» | Reconnection Address |

*Table 12.5: Reconnection Address characteristic*

The Reconnection Address characteristic value shall be 6 octets in length. A device shall have only one instance of the Reconnection Address characteristic.

## 12.5 PERIPHERAL PREFERRED CONNECTION PARAMETERS CHARACTERISTIC

The Peripheral Preferred Connection Parameters (PPCP) characteristic contains the preferred connection parameters of the Peripheral.

| Attribute Handle | Attribute Type | Attribute Value |
|---|---|---|
| 0xMMMM | 0x0x2A04 – UUID for «Peripheral Preferred Connection Parameters» | Peripheral Preferred Connection Parameter |

*Table 12.6: Peripheral Preferred Connection Parameters characteristic*

The Peripheral Preferred Connection Parameters characteristic value shall be 8 octets in length. A device shall have only one instance of the Peripheral Preferred Connection Parameters characteristic.

The preferred connection parameters structured data is defined as follows:

| Name | Size (Octet) | Description |
|---|---|---|
| Minimum connection interval | 2 | Defines minimum value for the connection interval in the following manner:<br>$connInterval_{min}$ = Conn_Interval_Min * 1.25 ms<br>Conn_Interval_Min range: 0x0006 to 0x0C80<br>Value of 0xFFFF indicates no specific minimum.<br>Values outside the range are reserved. (excluding 0xFFFF) |

*Table 12.7: Format of the preferred connection parameters structured data*

| Name | Size (Octet) | Description |
|---|---|---|
| Maximum connection interval | 2 | Defines maximum value for the connection interval in the following manner:<br>$connInterval_{max}$ = Conn_Interval_Max * 1.25 ms<br>Conn_Interval_Max range: 0x0006 to 0x0C80<br>Shall be equal to or greater than the Conn_Interval_Min.<br>Value of 0xFFFF indicates no specific maximum.<br>Values outside the range are reserved. (excluding 0xFFFF) |
| Slave latency | 2 | Defines the slave latency for the connection in number of connection events.<br>Slave latency range: 0x0000 to 0x03E8<br>Values outside the range are reserved. |
| Connection Supervision timeout multiplier | 2 | Defines the connection supervisor timeout multiplier as a multiple of 10ms. Range: 0xFFFF indicates no specific value requested.<br>Range: 0x000A to 0x0C80<br>Time = N * 10 msec<br>Time Range: 100 msec to 32 seconds<br>Values outside the range are reserved. (excluding 0xFFFF) |

*Table 12.7: Format of the preferred connection parameters structured data*

# 13 BR/EDR/LE OPERATION MODES AND PROCEDURE

This section describes the requirements for devices that support the BR/EDR/LE device type. The device may support any LE GAP roles allowed by the Controller over the LE physical channel.

| Feature | Ref. | Broadcaster | Observer | Peripheral | Central |
|---|---|---|---|---|---|
| BR/EDR/LE modes and procedures | 13.1 and 13.2 | O | O | O | O |

*Table 13.1: Requirements for the modes of a BR/EDR/LE device type*

A BR/EDR/LE device type that supports the Broadcaster role shall meet the requirements for the Broadcaster role as defined in Section 9.

A BR/EDR/LE device type that supports the Observer role shall meet the requirements for the Observer role as defined in Section 9.

A BR/EDR/LE device type that supports the Peripheral role shall meet the requirements for the Peripheral role as defined in Section 13.1, Section 13.2 and Section 13.3.

A BR/EDR/LE device type that supports the Central role shall meet the requirements for the Central role as defined in Section 13.1, Section 13.2 and Section 13.3.

## 13.1 MODES

The requirements for a device supporting BR/EDR/LE device type are shown in Table 13.2.

| Modes | Ref. | Peripheral | Central |
|---|---|---|---|
| Non-discoverable mode | 13.1.1.1 | C1 | C1 |
| Discoverable mode | 13.1.1.2 | C2 | C2 |
| Non-connectable mode | 13.1.2.1 | C3 | C3 |
| Connectable mode | 13.1.2.2 | M | M |
| Non-bondable mode | 13.1.3.1 | C4 | C4 |
| Bondable mode | 13.1.3.2 | C5 | C5 |

*Table 13.2: Requirements for the modes of a BR/EDR/LE device type*

| Modes | Ref. | Peripheral | Central |
|---|---|---|---|
| C1: Mandatory if the non-discoverable mode is supported over BR/EDR, else excluded | | | |
| C2: Mandatory if the discoverable mode is supported over BR/EDR, else excluded | | | |
| C3: Mandatory if the non-connectable mode is supported over BR/EDR, else excluded: | | | |
| C4: Mandatory if the non-bondable mode is supported over BR/EDR, else excluded | | | |
| C5: Mandatory if the bondable mode is supported over BR/EDR, else excluded | | | |

*Table 13.2: Requirements for the modes of a BR/EDR/LE device type*

### 13.1.1 Discoverability

#### *13.1.1.1 Non-Discoverable Mode*

A device in non-discoverable mode shall follow the requirements for non-discoverable mode for BR/EDR as defined in Section 4.1.1 and it shall follow the requirements for non-discoverable mode for LE as defined in Section 9.2.2

#### *13.1.1.2 Discoverable Mode*

A device in general discoverable mode shall follow the requirements for general discoverable mode for BR/EDR as defined in Section 4.1.3 and it shall follow the requirements for general discoverable mode for LE as defined in Section 9.2.4, except it shall not send connectable advertising events.

A device in limited discoverable mode shall follow the requirements for limited discoverable mode for BR/EDR as defined in Section 4.1.2 and it shall follow the requirements for limited discoverable mode for LE as defined in Section 9.2.3, except it shall not send connectable advertising events.

A device that supports general discoverable and limited discoverable modes concurrently on different physical channels is not permitted in this version of the specification.

A device supporting both BR/EDR and LE physical links shall expose the capabilities of both physical links by performing the following steps:

a) The ‘LE Supported (Controller)’ and ‘LE Supported (Host)’ bits in the LMP features shall be set as defined in [Vol. 2], Part C Section 3.2.

b) The ‘BR/EDR Not Supported’ bit in the Flags AD type shall be set to ‘0’ as defined in Section 18.1.

c) The ‘Simultaneous LE and BR/EDR to Same Device Capable (Controller)’ and ‘Simultaneous LE and BR/EDR to Same Device

Capable (Host)' bits in the Flags AD type shall be set to '0' as defined in Section 18.1.

### 13.1.2 Connectability

#### *13.1.2.1 Non-Connectable Mode*

A device in the non-connectable mode shall follow the requirements for the non-connectable mode for BR/EDR as defined in Section 4.2.1 and it shall follow the requirements for the non-connectable mode for LE as defined in Section 9.3.2.

#### *13.1.2.2 Connectable Mode*

A device in the connectable mode shall follow the requirements for the connectable mode for BR/EDR as defined in Section 4.2.2, and it shall follow the requirements for the non-connectable mode for LE as defined in Section 9.3.2. The Host shall not set the "Simultaneous LE and BR/EDR to same device capable (Host)" LMP feature bit as defined in [Vol. 2], Part C Section 3.2.

### 13.1.3 Bondable Modes

#### *13.1.3.1 Non-Bondable Mode*

A device in the non-bondable mode shall follow the requirements for the non-bondable mode for BR/EDR as defined in Section 4.3.1 and it shall follow the requirements for the non-bondable mode for LE as defined in Section 9.4.2.

#### *13.1.3.2 Bondable Mode*

For a device that is operating in the LE Central role, a device in the bondable mode shall follow the requirements for the bondable mode for BR/EDR as defined inSection 4.3.2 and it shall follow the requirements for the bondable mode for LE as defined in Section 9.4.3.

For a device that is operating in the LE Peripheral role, a device in the bondable mode shall follow the requirements for the bondable mode for BR/EDR as defined in Section 4.3.2 and it shall follow the requirements for the non-bondable mode for LE as defined in Section 9.4.2.

## 13.2 IDLE MODE PROCEDURES

The requirements for a device supporting BR/EDR/LE device type are shown in Table 13.3.

| Procedures | Ref. | Peripheral | Central |
|---|---|---|---|
| General discovery procedure | 13.2.1 | E | C1 |
| Limited discovery procedure | 13.2.2 | E | C2 |
| Device type discovery procedure | 13.2.3.1 | E | C3 |
| Name discovery procedure | 13.2.4 | E | C4 |
| Link establishment procedure | 13.3 | E | C5 |

C1: Mandatory if the general discovery procedure is supported over BR/EDR, else excluded

C2: Mandatory if the limited discovery procedure is supported over BR/EDR, else excluded

C3: Mandatory if either the general discovery or the limited discovery procedure is supported over BR/EDR, else excluded

C4: Mandatory if the name discovery procedure is supported over BR/EDR, else excluded

C5: Mandatory if the link establishment procedure is supported over BR/EDR, else excluded

*Table 13.3: Requirements for the procedures of a BR/EDR/LE device type*

### 13.2.1 General Discovery Procedure

In order to provide a good user experience for discovering remote devices, the general discovery procedure should interleave the general discovery procedure over BR/EDR as defined in Section 6.1 with the general discovery procedure over LE as defined in Section 9.2.6. A recommended method for interleaving is:

Step 1: perform the general discovery procedure over BR/EDR for ($T_{GAP}$(100) / 2) seconds

Step 2: perform the general discovery procedure over LE for ($T_{GAP}$(100) / 2) seconds

Step 3: repeat step 1 and 2 as necessary

### 13.2.2 Limited Discovery Procedure

In order to provide a good user experience for discovering remote devices, the limited discovery procedure should interleave the limited discovery procedure over BR/EDR as defined in Section 6.2 with the limited discovery procedure

over LE as defined in Section 9.2.5. A recommended method for interleaving is:

Step 1: perform the limited discovery procedure over BR/EDR for ($T_{GAP}$(100) / 2) seconds

Step 2: perform the limited discovery procedure over LE for ($T_{GAP}$(100) / 2) seconds

Step 3: repeat step 1 and 2 as necessary

### 13.2.3 Device Discovery Procedure

A discovering device shall be capable of discovering devices of each of the following device types:

- BR/EDR
- LE-only
- BR/EDR/LE

The device discovery procedure shall perform either the general discovery procedure as defined in Section 13.2.1 or the limited discovery procedure as defined in Section 13.2.2. The device type discovery procedure as defined in Section 13.2.3.1 shall be used to determine if the remote device supports both BR/EDR and LE physical links. Using the information from the device type discovery procedure a name discovery may be performed for each device discovered as defined in Section 13.2.4.

#### *13.2.3.1 Device Type Discovery Procedure*

The device type discovery procedure can be performed to discover the device type of the peer device.

If the procedure is performed on a BR/EDR physical link:

If the ‘LE Supported (Controller)’ bit in the LMP features is set and the ‘LE Supported (Host)’ bit in the LMP host features is set, then the device supports the BR/EDR/LE device type, else the device supports the BR/EDR device type.

If the procedure is performed on an LE physical channel:

If the ‘BR/EDR Not Supported’ bit in the Flags AD type is set then the device type is LE-only, else the device type is BR/EDR/LE. The device type shall not change while the device is bonded with other devices.

### 13.2.4 Name Discovery

When the remote device is an LE-only device type, the local device shall use the name discovery procedure as defined in Section 13.2.4. When the remote

device type is BR/EDR or BR/EDR/LE device type, the local device shall use the name discovery procedure as defined in Section 6.3.

Note: The device type of the remote device can be discovered using the device type discovery procedure as defined in Section 13.2.3.1.

## 13.3 ESTABLISHMENT PROCEDURES

### 13.3.1 Link Establishment

When establishing a link to a remote device that supports the BR/EDR/LE device type, the link establishment procedure as defined in Section 7.1 shall be used, and shall not use the connection establishment procedures as defined in Section 9.3.

When establishing a link to a remote device that supports the BR/EDR device type, the link establishment procedure as defined in Section 7.1 shall be used.

When establishing a link to a remote device of the LE-only device type, the connection establishment procedures as defined in Section 9.3 shall be used.

Note: The device type of the remote device can be discovered using the device type discovery procedure as defined in Section 13.2.3.1.

## 13.4 SDP INTEROPERABILITY REQUIREMENTS

A BR/EDR/LE device that supports GATT over BR/EDR shall publish the following SDP record. The GAP Start Handle shall be set to the attribute handle of the «Generic Access Profile» service declaration. The GAP End Handle shall be set to the attribute handle of the last attribute within the «Generic Access Profile» service definition group.

| Item | Definition | Type | Value | Status |
|---|---|---|---|---|
| Service Class ID List | | | | M |
| Service Class #0 | | UUID | Generic Access Profile | M |
| Protocol Descriptor List | | | | M |
| Protocol #0 | | UUID | L2CAP | M |
| Parameter #0 for Protocol #0 | PSM | Uint16 | PSM = ATT | M |
| Protocol #1 | | UUID | ATT | M |

*Table 13.4: SDP Record for the Generic Access Profile*

| Item | Definition | Type | Value | Status |
|---|---|---|---|---|
| Parameter #0 for Protocol #1 | GAP Start Handle | Uint16 | | M |
| Parameter #1 for Protocol #1 | GAP End Handle | Uint16 | | M |
| BrowseGrou-pList | | | PublicBrowseGroup | M |

*Table 13.4: SDP Record for the Generic Access Profile*

# 14 BR/EDR/LE SECURITY ASPECTS

The requirements for a device supporting BR/EDR/LE device type are shown in Table 14.1.

| Security aspects | Ref. | Peripheral | Central |
|---|---|---|---|
| Security aspects | 14 | M | M |

*Table 14.1: Requirements for the security aspects of a BR/EDR/LE device type*

If the remote device supports the BR/EDR/LE device type or the BR/EDR device type the security procedures as defined in Section 5 shall be used.

If the remote device is of the LE-only device type the security procedures as defined in Section 10 shall be used.

Note: The device type of the remote device can be discovered using the device type discovery procedure as defined in Table 13.2.3.1.

# 15 DEFINITIONS

In the following, terms written with capital letters refer to states.

## 15.1 GENERAL DEFINITIONS

**Mode .** A set of directives that defines how a device will respond to certain events.

**Idle .** As seen from a remote device, a Bluetooth device is idle, or is in idle mode, when there is no link established between them.

**Bond .** A relation between two Bluetooth devices defined by creating, exchanging and storing a common link key. The bond is created through the bonding or LMP-pairing procedures.

## 15.2 CONNECTION-RELATED DEFINITIONS

**Physical channel .** A synchronized Bluetooth baseband-compliant RF hopping sequence.

**Piconet .** A set of Bluetooth devices sharing the same physical channel defined by the master parameters (clock and BD_ADDR).

**Physical link .** A Baseband-level connection[1] between two devices established using paging. A physical link comprises a sequence of transmission slots on a physical channel alternating between master and slave transmission slots.

**ACL link .** An asynchronous (packet-switched) connection[1] between two devices created on LMP level. Traffic on an ACL link uses ACL packets to be transmitted.

**SCO link .** A synchronous (circuit-switched) connection[1] for reserved bandwidth communications; e.g. voice between two devices, created on the LMP level by reserving slots periodically on a physical channel. Traffic on an SCO link uses SCO packets to be transmitted. SCO links can be established only after an ACL link has first been established.

**Link .** Shorthand for an ACL link.

**PAGE .** A baseband state where a device transmits page trains, and processes any eventual responses to the page trains.

**PAGE_SCAN .** A baseband state where a device listens for page trains.

1. The term 'connection' used here is not identical to the definition below. It is used in the absence of a more concise term.

**Page .** The transmission by a device of page trains containing the Device Access Code of the device to which the physical link is requested.

**Page scan .** The listening by a device for page trains containing its own Device Access Code.

**Channel .** A logical connection on L2CAP level between two devices serving a single application or higher layer protocol.

**Connection .** A connection between two peer applications or higher layer protocols mapped onto a channel.

**Connecting .** A phase in the communication between devices when a connection between them is being established. (Connecting phase follows after the link establishment phase is completed.)

**Connect (to service) .** The establishment of a connection to a service. If not already done, this includes establishment of a physical link, link and channel as well.

## 15.3 DEVICE-RELATED DEFINITIONS

**Discoverable device .** A Bluetooth device in range that will respond to an inquiry (normally in addition to responding to page).

**Silent device .** A Bluetooth device appears as silent to a remote device if it does not respond to inquiries made by the remote device. A device may be silent due to being non-discoverable or due to baseband congestion while being discoverable.

**Connectable device .** A Bluetooth device in range that will respond to a page.

**Trusted device .** A paired device that is explicitly marked as trusted.

**Paired device .** A Bluetooth device with which a link key has been exchanged (either before connection establishment was requested or during connecting phase).

**Pre-paired device .** A Bluetooth device with which a link key was exchanged, and the link key is stored, before link establishment.

**Un-paired device .** A Bluetooth device for which there was no exchanged link key available before connection establishment was requested.

**Known device .** A Bluetooth device for which at least the BD_ADDR is stored.

**Un-known device .** A Bluetooth device for which no information (BD_ADDR, link key or other) is stored.

**Authenticated device .** A Bluetooth device whose identity has been verified during the lifetime of the current link, based on the authentication procedure.

## 15.4 PROCEDURE-RELATED DEFINITIONS

**Paging.** A procedure for establishing a physical link of ACL type on baseband level, consisting of a page action of the initiator and a page scan action of the responding device.

**Link establishment .** A procedure for establishing a link on LMP level. A link is established when both devices have agreed that LMP setup is completed.

**Channel establishment .** A procedure for establishing a channel on L2CAP level.

**Connection establishment .** A procedure for creating a connection mapped onto a channel.

**Creation of a trusted relationship .** A procedure where the remote device is marked as a trusted device. This includes storing a common link key for future authentication and pairing (if the link key is not available).

**Creation of a secure connection.** A procedure of establishing a connection, including authentication and encryption.

**Device discovery .** A procedure for retrieving the Bluetooth device address, clock, class-of-device field and used page scan mode from discoverable devices.

**Name discovery .** A procedure for retrieving the user-friendly name (the Bluetooth device name) of a connectable device.

**Service discovery .** Procedures for querying and browsing for services offered by or through another Bluetooth device.

## 15.5 SECURITY-RELATED DEFINITIONS

**Authentication .** A generic procedure based on LMP-authentication if a link key exists or on LMP-pairing if no link key exists.

**LMP-authentication .** An LMP level procedure for verifying the identity of a remote device. The procedure is based on a challenge-response mechanism using a random number, a secret key and the BD_ADDR of the non-initiating device. The secret key used can be a previously exchanged link key.

**Authorization .** A procedure where a user of a Bluetooth device grants a specific (remote) Bluetooth device access to a specific service. Authorization implies that the identity of the remote device can be verified through authentication.

**Authorize .** The act of granting a specific Bluetooth device access to a specific service. It may be based upon user confirmation, or given the existence of a trusted relationship.

**LMP-pairing .** A procedure that authenticates two devices, based on a PIN, and subsequently creates a common link key that can be used as a basis for a trusted relationship or a (single) secure connection. The procedure consists of the steps: creation of an initialization key (based on a random number and a PIN), creation and exchange of a common link key and LMP-authentication based on the common link key.

**Bonding .** A dedicated procedure for performing the first authentication, where a common link key is created and stored for future use.

**Trusting .** The marking of a paired device as trusted. Trust marking can be done by the user, or automatically by the device (e.g. when in bondable mode) after a successful pairing.

# 16 APPENDIX A (NORMATIVE): TIMERS AND CONSTANTS

The following timers are required by this profile.

| Timer name | Value | Description | Requirement or Recommendation |
|---|---|---|---|
| $T_{GAP}(100)$ | 10.24 s | Time span that a Bluetooth device performs device discovery. | Recommended value |
| $T_{GAP}(101)$ | 10.625 ms | A discoverable Bluetooth device enters INQUIRY_SCAN for at least TGAP(101) every TGAP(102). | Required value |
| $T_{GAP}(102)$ | 2.56 s | Maximum time between repeated INQUIRY_SCAN enterings. | Recommended value |
| $T_{GAP}(103)$ | 30.72 s | Minimum time span that a device is in discoverable mode. | Required value |
| $T_{GAP}(104)$ | 1 min. | Maximum time span that a device is in limited discoverable mode. | Recommended value |
| $T_{GAP}(105)$ | 100ms | Maximum time between INQUIRY_SCAN enterings | Recommended value |
| $T_{GAP}(106)$ | 100ms | Maximum time between PAGE_SCAN enterings | Recommended value |
| $T_{GAP}(107)$ | 1.28s | Maximum time between PAGE_SCAN enterings (R1 page scan) | Recommended value |
| $T_{GAP}(108)$ | 2.56s | Maximum time between PAGE_SCAN enterings (R2 page scan) | Recommended value |
| $T_{GAP}(lim\_adv\_timeout)$ | 30.72 s | Maximum time to remain advertising when in the limited discoverable mode | Required value |
| $T_{GAP}(gen\_disc\_scan\_min)$ | 10.24 s | Minimum time to perform scanning when performing the general discovery procedure | Recommended value |

*Table 16.1: Defined GAP timers*

| Timer name | Value | Description | Requirement or Recommendation |
|---|---|---|---|
| $T_{GAP}$(gen_disc_scan_int) | 11.25 ms | Scan interval used in the general discovery procedure | Recommended value |
| $T_{GAP}$(gen_disc_scan_wind) | 11.25 ms | Scan window used in the general discovery procedure | Recommended value |
| $T_{GAP}$(lim_disc_scan_min) | 10.24 s | Minimum time to perform scanning when performing the limited discovery procedure | Recommended value |
| $T_{GAP}$(lim_disc_scan_int) | 11.25 ms | Scan interval used in the limited discovery procedure | Recommended value |
| $T_{GAP}$(lim_disc_scan_wind) | 11.25 ms | Scan window used in the limited discovery procedure | Recommended value |
| $T_{GAP}$(conn_param_timeout) | 30 s | Connection parameter update notification timer when performing the connection parameter update procedure | Recommended value |
| $T_{GAP}$(conn_est_int_min) | 500 ms | Minimum connection interval in any connection establishment procedure | Recommended value |
| $T_{GAP}$(conn_est_int_max) | 1.28 s | Maximum connection interval in any connection establishment procedure | Recommended value |
| $T_{GAP}$(conn_est_latency) | 0 | Connection slave latency in any connection establishment procedure | Recommended value |
| $T_{GAP}$(private_addr_int) | 15 mins | Minimum time interval between private address change | Recommended value |
| $T_{GAP}$(lim_disc_adv_int_min) | 250 ms | Minimum advertising interval when in the limited discoverable mode | Recommended value |
| $T_{GAP}$(lim_disc_adv_int_max) | 500 ms | Maximum advertising interval when in the limited discoverable mode | Recommended value |
| $T_{GAP}$(gen_disc_adv_int_min) | 1.28 s | Minimum advertising interval when in the general discoverable mode | Recommended value |

*Table 16.1: Defined GAP timers*

| Timer name | Value | Description | Requirement or Recommendation |
|---|---|---|---|
| $T_{GAP}$(gen_disc_adv_int_max) | 2.56 s | Maximum advertising interval when in the general discoverable mode | Recommended value |
| $T_{GAP}$(dir_conn_adv_int_min) | 250 ms | Minimum advertising interval when in the directed connectable mode | Recommended value |
| $T_{GAP}$(dir_conn_adv_int_max) | 500 ms | Maximum advertising interval when in the directed connectable mode | Recommended value |
| $T_{GAP}$(undir_conn_adv_int_min) | 1.28 s | Minimum advertising interval when in the undirected connectable mode | Recommended value |
| $T_{GAP}$(undir_conn_adv_int_max) | 2.56 s | Maximum advertising interval when in the undirected connectable mode | Recommended value |
| $T_{GAP}$(auto_conn_est_scan_int) | 1.28 s | Scan interval used in the auto connection establishment procedure | Recommended value |
| $T_{GAP}$(auto_conn_est_scan_wind) | 11.25 ms | Scan window used in the auto connection establishment procedure | Recommended value |
| $T_{GAP}$(gen_conn_est_scan_int) | 1.28 s | Scan interval used in the general connection establishment procedure | Recommended value |
| $T_{GAP}$(gen_conn_est_scan_wind) | 11.25 ms | Scan window used in the general connection establishment procedure | Recommended value |
| $T_{GAP}$(sel_conn_est_scan_int) | 1.28 s | Scan interval used in the selective connection establishment procedure | Recommended value |
| $T_{GAP}$(sel_conn_est_scan_wind) | 11.25 ms | Scan window used in the selective connection establishment procedure | Recommended value |
| $T_{GAP}$(conn_param_timeout) | 30 s | Timer used in connection parameter update procedure | Recommended value |

*Table 16.1: Defined GAP timers*

# 17 APPENDIX B (INFORMATIVE): INFORMATION FLOWS OF RELATED PROCEDURES

## 17.1 LMP – AUTHENTICATION

The specification of authentication on link level is found in [2].

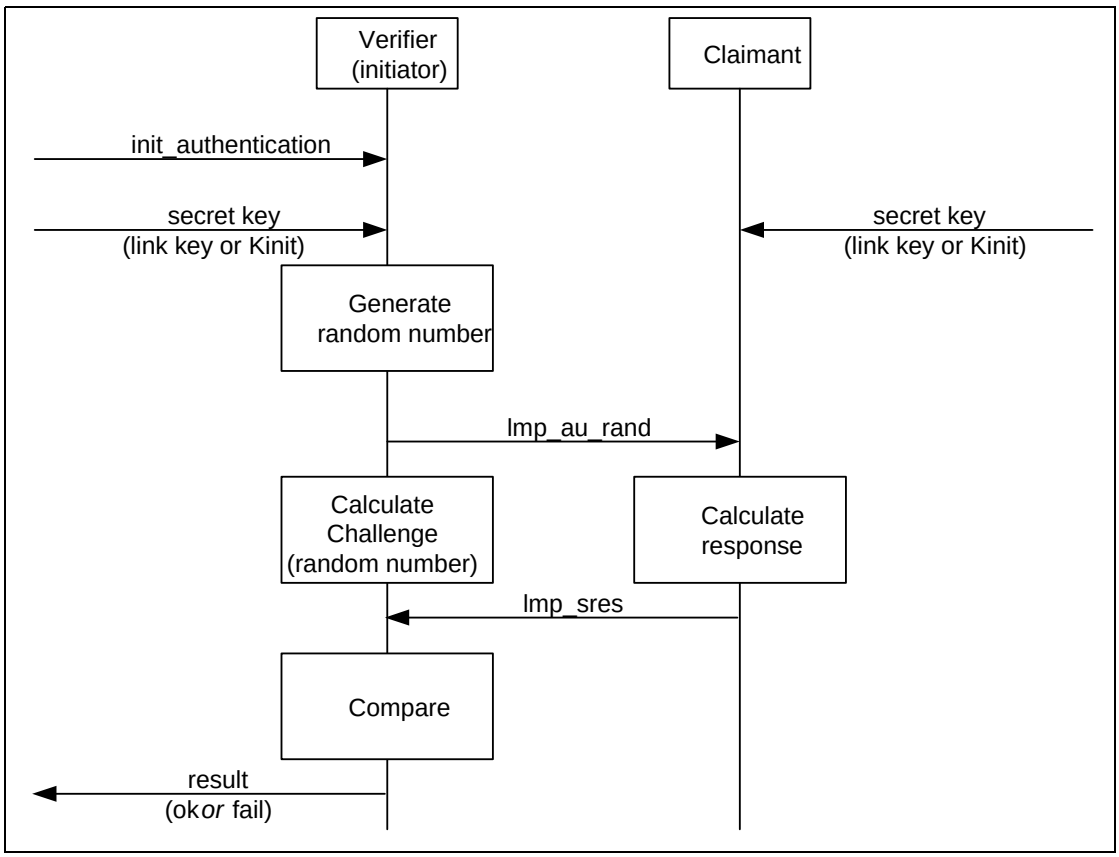



*Figure 17.1: LMP-authentication as defined by [2].*

The secret key used here is an already exchanged link key.

## 17.2 LMP – PAIRING

The specification of pairing on link level is found in [2].

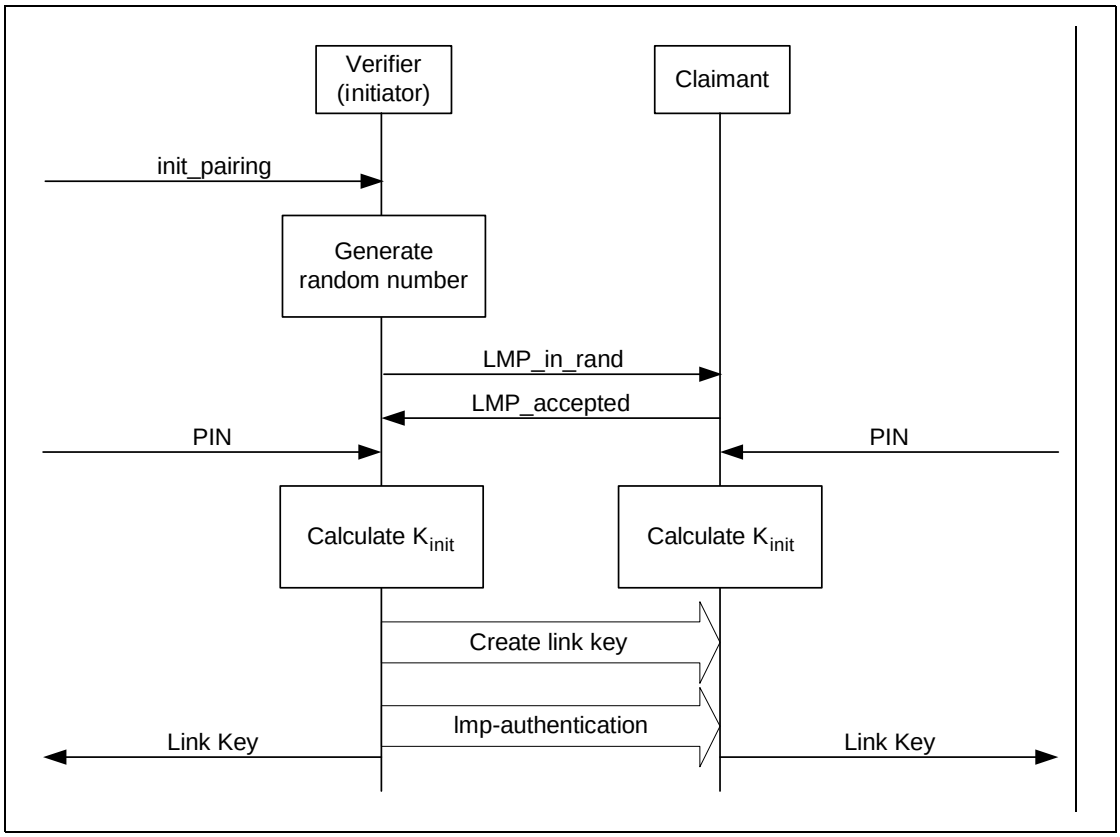



*Figure 17.2: LMP-pairing as defined in [2].*

The PIN used here is PINBB.

The create link key procedure is described in [Vol. 2, Part C] Section 4.2.2.4 on page 251 and [Vol. 2, Part H] Section 3.2 on page 1065. In case the link key is based on a combination key, a mutual authentication takes place and shall be performed irrespective of current security mode.

## 17.3 SERVICE DISCOVERY

The Service Discovery Protocol [5] specifies what PDUs are used over-the-air to inquire about services and service attributes. The procedures for discovery of supported services and capabilities using the Service Discovery Protocol are described in higher layer specifications. This is just an example.

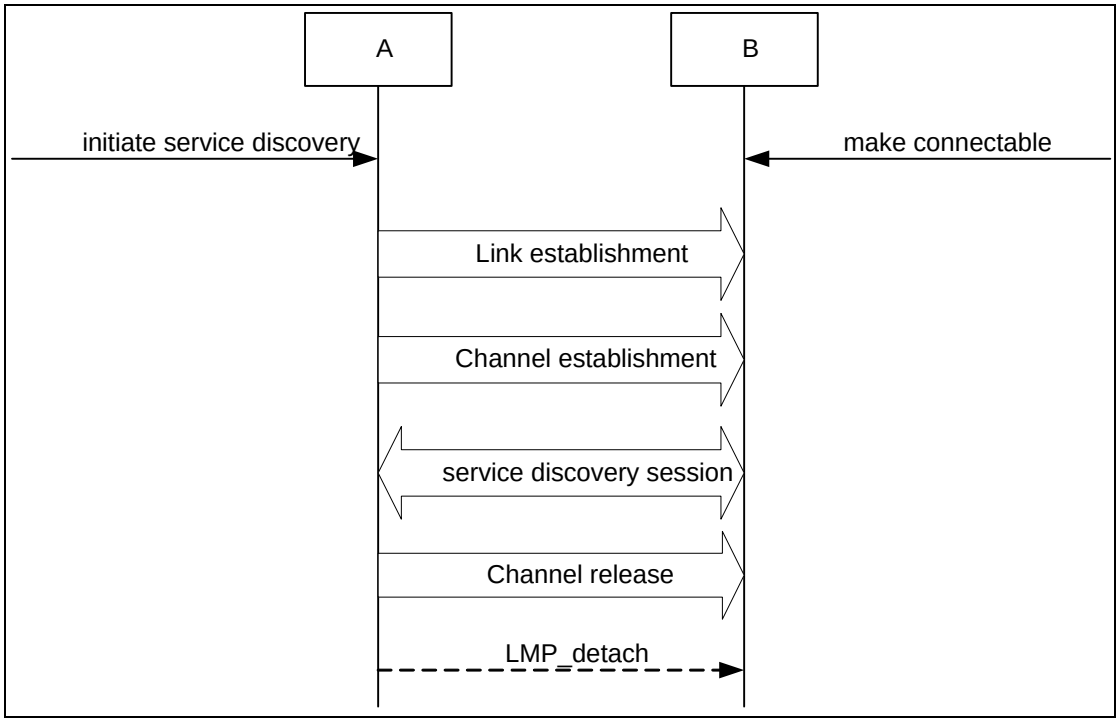



*Figure 17.3: Service discovery procedure.*

## 17.4 GENERATING A RESOLVABLE PRIVATE ADDRESS

Generating a resolvable private address is described in Section 10.8.2.2.

## 17.5 RESOLVING A RESOLVABLE PRIVATE ADDRESS

Resolving a resolvable private address is described in Section 10.8.2.3.

# 18 APPENDIX C (NORMATIVE): EIR AND AD FORMATS

This section defines the data format used in the EIR data field and in the AD format.

## 18.1 FLAGS

The Flags field may be zero or more octets long. This allows the Flags field to be extended while using the minimum number of octets within the data packet. All octets that are 0x00 are not transmitted as long as all other octets after that octet are also 0x00.

| Value | Description | Bit | Information |
|---|---|---|---|
| 0x01 | Flags | 0 | LE Limited Discoverable Mode |
| | | 1 | LE General Discoverable Mode |
| | | 2 | BR/EDR Not Supported (i.e. bit 37 of LMP Extended Feature bits Page 0) |
| | | 3 | Simultaneous LE and BR/EDR to Same Device Capable (Controller) (i.e. bit 49 of LMP Extended Feature bits Page 0) |
| | | 4 | Simultaneous LE and BR/EDR to Same Device Capable (Host) (i.e. bit 66 of LMP Extended Feature bits Page 1) |
| | | 5..7 | Reserved |

*Table 18.1: Flags*

## 18.2 SERVICE

| Value | Description | Information |
|---|---|---|
| 0x02 | 16-bit Service UUIDs | More 16-bit UUIDs available |
| 0x03 | 16-bit Service UUIDs | Complete list of 16-bit UUIDs available |
| 0x04 | 32-bit Service UUIDs | More 32-bit UUIDs available |
| 0x05 | 32-bit Service UUIDs | Complete list of 32-bit UUIDs available |
| 0x06 | 128-bit Service UUIDs | More 128-bit UUIDs available |
| 0x07 | 128-bit Service UUIDs | Complete list of 128-bit UUIDs available |

*Table 18.2: Service UUIDs*

## 18.3 LOCAL NAME

| Value | Description | Information |
|---|---|---|
| 0x08 | Local Name | Shortened local name |
| 0x09 | Local Name | Complete local name |

*Table 18.3: Local Name*

## 18.4 TX POWER LEVEL

| Value | Description | Information |
|---|---|---|
| 0x0A | TX Power Level (1 byte) | 0xXX:-127 to +127dBm<br>Note: when the TX Power Level tag is not present, the TX power level of the packet is unknown. |

*Table 18.4: TX Power Level*

## 18.5 SIMPLE PAIRING OPTIONAL OOB TAGS

| Value | Description | Information |
|---|---|---|
| 0x0D | Class of device (3 octets) | Format defined in Assigned Numbers |
| 0x0E | Simple Pairing Hash C (16 octets) | Format defined in [Vol. 2], Part H Section 7.2.2 |
| 0x0F | Simple Pairing Randomizer R (16 octets) | Format defined in[Vol. 2], Part H Section 7.2.2 |

*Table 18.5: Optional OOB Tags*

## 18.6 SECURITY MANAGER TK VALUE

| Value | Description | Information |
|---|---|---|
| 0x10 | TK Value | Value as used in pairing over LE Physical channel.<br>Format defined in [Vol. 3], Part H Section 2.3 |

*Table 18.6: Security Manager TK Value*

## 18.7 SECURITY MANAGER OOB FLAGS

| Value | Description | Bit | Information |
|---|---|---|---|
| 0x11 | Flag (1 octet) | 0 | OOB Flags Field<br>(0 = OOB data not present, 1 = OOB data present) |
| | | 1 | LE supported (Host) (i.e. bit 65 of LMP Extended Feature bits Page 1 |
| | | 2 | Simultaneous LE and BR/EDR to Same Device Capable (Host) (i.e. bit 66 of LMP Extended Feature bits Page 1) |
| | | 3 | Address type (0 = Public Address, 1 = Random Address) |
| | | 4.7 | Reserved |

*Table 18.7: Security Manager OOB Flags*

## 18.8 SLAVE CONNECTION INTERVAL RANGE

| Value | Description | Information |
|---|---|---|
| 0x12 | Slave Connection Interval Range | The first 2 octets defines the minimum value for the connection interval in the following manner:<br>$connInterval_{min}$ = Conn_Interval_Min * 1.25 ms<br>Conn_Interval_Min range: 0x0006 to 0x0C80<br>Value of 0xFFFF indicates no specific minimum.<br>Values outside the range are reserved. (excluding 0xFFFF)<br><br>The second 2 octets defines the maximum value for the connection interval in the following manner:<br><br>$connInterval_{max}$ = Conn_Interval_Max * 1.25 ms<br>Conn_Interval_Max range: 0x0006 to 0x0C80<br>Conn_Interval_Max shall be equal to or greater than the Conn_Interval_Min.<br>Value of 0xFFFF indicates no specific maximum.<br>Values outside the range are reserved (excluding 0xFFFF) |

*Table 18.8: Slave Connection Interval Range*

## 18.9 SERVICE SOLICITATION

| Value | Description | Information |
|---|---|---|
| 0x14 | Service UUIDs | List of 16 bit Service UUIDs |
| 0x15 | Service UUIDs | List of 128 bit Service UUID |

*Table 18.9: Service Solicitation*

## 18.10 SERVICE DATA

| Value | Description | Information |
|---|---|---|
| 0x16 | Service Data (2 or more octets) | The first 2 octets contain the 16 bit Service UUID followed by additional service data |

*Table 18.10: Service Data*

## 18.11 MANUFACTURER SPECIFIC DATA

| Value | Description | Information |
|---|---|---|
| 0xFF | Manufacturer Specific Data (2 or more octets) | The first 2 octets contain the Company Identifier Code followed by additional manufacturer specific data |

*Table 18.11: Manufacturer Specific Data*

# 19 REFERENCES

[1] Baseband Specification

[2] Link Manager Protocol

[3] RFCOMM

[4] Telephony Control Specification

[5] Service Discovery Protocol

[6] Security Architecture (white paper)

[7] Bluetooth Assigned Numbers

# TEST SUPPORT

# CONTENTS

**1 Test Methodology ..... 411**
1.1 BR/EDR Test Scenarios ..... 411
1.1.1 Test setup ..... 411
1.1.2 Transmitter Test ..... 412
1.1.2.1 Packet Format ..... 413
1.1.2.2 Pseudorandom Sequence ..... 414
1.1.2.3 Control of Transmit Parameters ..... 414
1.1.2.4 Power Control ..... 415
1.1.2.5 Switch Between Different Frequency Settings ..... 415
1.1.2.6 Adaptive Frequency Hopping ..... 416
1.1.3 LoopBack test ..... 416
1.1.4 Pause test ..... 420
1.2 AMP Test Scenarios ..... 421
1.2.1 Methodology Overview ..... 421
1.2.1.1 Initiation Example Description ..... 422
1.2.2 Control and Configuration ..... 422
1.2.3 AMP Test Manager ..... 423
1.2.4 Test Commands/Events Format ..... 424
1.2.5 AMP Test Manager Commands/Events ..... 425
1.2.5.1 AMP Command Rejected Event ..... 426
1.2.5.2 AMP Discover Request ..... 427
1.2.5.3 AMP Discover Response Event ..... 427
1.2.5.4 AMP Read PHY Capability Bit Map Command ..... 428
1.2.5.5 AMP Read PHY Capability Bit Map Response Event ..... 429
1.3 References ..... 429
**2 Test Control Interface (TCI) ..... 430**
2.1 Introduction ..... 430
2.1.1 Terms Used ..... 430
2.1.2 Usage of the Interface ..... 430
2.2 TCI Configurations ..... 431
2.2.1 Bluetooth RF Requirements ..... 431
2.2.1.1 Required interfaces ..... 431
2.2.2 Bluetooth protocol requirements ..... 432
2.2.2.1 Required interfaces ..... 432
2.2.3 Bluetooth Profile Requirements ..... 433
2.2.3.1 Required interfaces ..... 433

2.3 TCI Configuration and Usage ..... 434
2.3.1 Transport Layers ..... 434
2.3.1.1 Physical bearer ..... 434
2.3.1.2 Software bearer ..... 434
2.3.2 Baseband and Link Manager Qualification ..... 435
2.3.3 HCI Qualification ..... 437

# 1 TEST METHODOLOGY

This section describes the test modes for hardware and low-level functionality tests of Bluetooth devices.

The BR/EDR test mode supports testing of the Bluetooth transmitter and receiver including transmitter tests (packets with constant bit patterns) and loopback tests. It is intended mainly for certification/compliance testing of the radio and baseband layer, and may also be used for regulatory approval or in-production and after-sales testing.

For AMP, this section describes the AMP test mode for hardware and low-level functionality tests of Bluetooth AMP devices. The AMP test mode provides the ability to control an AMP device using HCI Commands for both MAC conformance testing and PHY transmitter and receiver tests utilising the BR/EDR connection. The AMP test mode is intended mainly for certification/compliance testing of the MAC and PHY and may also be used for regulatory approval or in production and after sales testing.

## 1.1 BR/EDR TEST SCENARIOS

A device in BR/EDR test mode shall not support normal operation. For security reasons the BR/EDR test mode is designed such that it offers no benefit to the user. Therefore, no data output or acceptance on a HW or SW interface shall be allowed.

### 1.1.1 Test setup

The setup consists of a device under test (DUT) and a tester. Optionally, additional measurement equipment may be used.

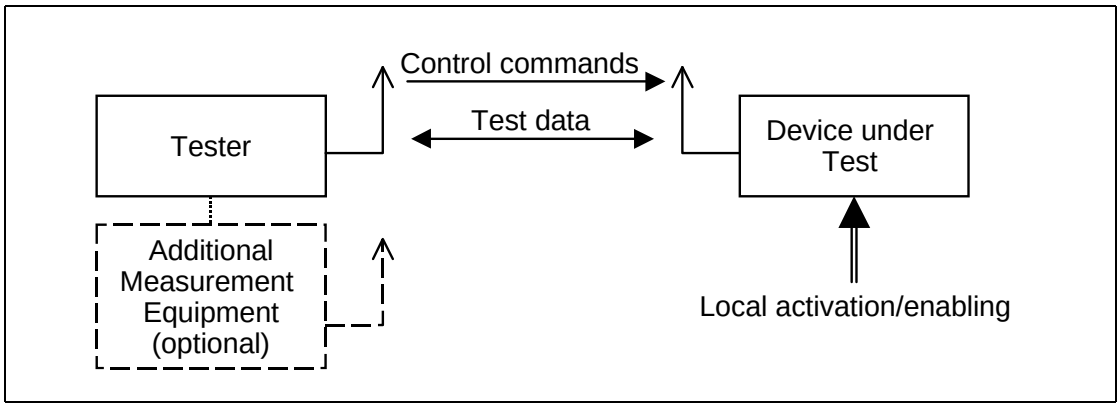



*Figure 1.1: Setup for Test Mode*

Tester and DUT form a piconet where the tester acts as master and has full control over the test procedure. The DUT acts as slave.

The control is done via the air interface using LMP commands (see [Vol. 2, Part C] Section 4.7.3 on page 306). Hardware interfaces to the DUT may exist, but are not subject to standardization.

The test mode is a special state of the Bluetooth model. For security and type approval reasons, a Bluetooth device in test mode shall not support normal operation. When the DUT leaves the test mode it enters the standby state. After power-off the Bluetooth device shall return to standby state.

### 1.1.2 Transmitter Test

The Bluetooth device transmits a constant bit pattern. This pattern is transmitted periodically with packets aligned to the slave TX timing of the piconet formed by tester and DUT. The same test packet is repeated for each transmission.

The transmitter test is started when the master sends the first POLL packet. In non-hopping mode agreed frequency is used for this POLL packet.

The tester (master) transmits control or POLL packets in the master-to-slave transmission slots. The DUT (slave) shall transmit packets in the following slave-to-master transmission slot. The tester's polling interval is fixed and defined by the LMP_test_control PDU. The device under test may transmit its burst according to the normal timing even if no packet from the tester was received. In this case, the ARQN bit is shall be set to NAK.

The burst length may exceed the length of a one slot packet. In this case the tester may take the next free master TX slot for polling. The timing is illustrated in Figure 1.2.

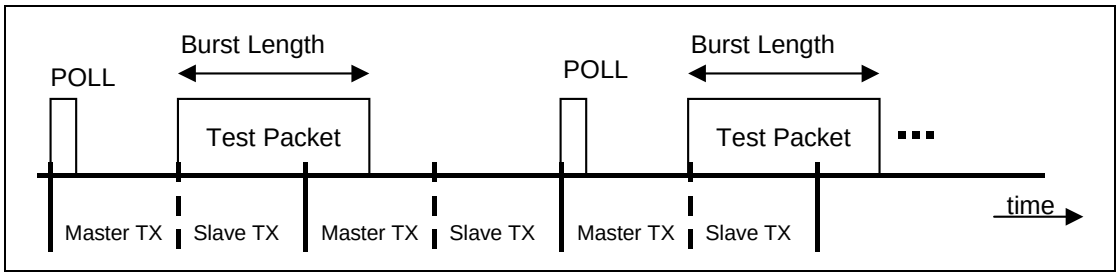



*Figure 1.2: Timing for Transmitter Test*

### *1.1.2.1 Packet Format*

The test packet is a normal Bluetooth packet, see Figure 1.3. For the payload itself see below.

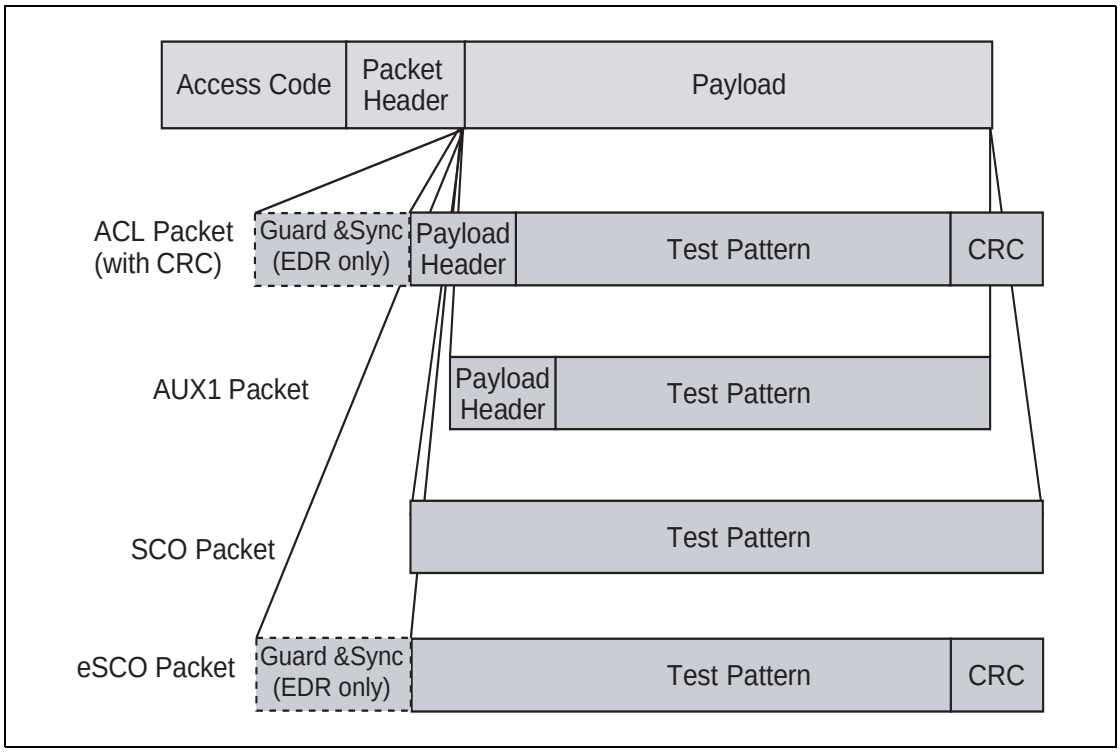



*Figure 1.3: General Format of TX Packet*

During configuration the tester defines:

- the packet type to be used
- payload length

For the payload length, the restrictions from the baseband specification shall apply (see "Baseband Specification" on page 59[vol. 2]). In case of ACL, SCO and eSCO packets the payload structure defined in the baseband specification is preserved as well, see Figure 1.3 on page 413.

For the transmitter test mode, only packets without FEC should be used; i.e. HV3, EV3, EV5, DH1, DH3, DH5, 2-EV3, 2-EV5, 3-EV3, 3-EV5, 2-DH1, 2-DH3, 2-DH5, 3-DH1, 3-DH3, 3-DH5 and AUX1 packets.

In transmitter test mode, the packets exchanged between the tester and the DUT shall not be scrambled with the whitening sequence. Whitening shall be turned off when the DUT has accepted to enter the transmitter test mode, and shall be turned on when the DUT has accepted to exit the transmitter test mode, see Figure 1.4 on page 414. Implementations shall ensure that retransmissions of the LMP_accepted messages use the same whitening status as used in the original LMP_accepted.

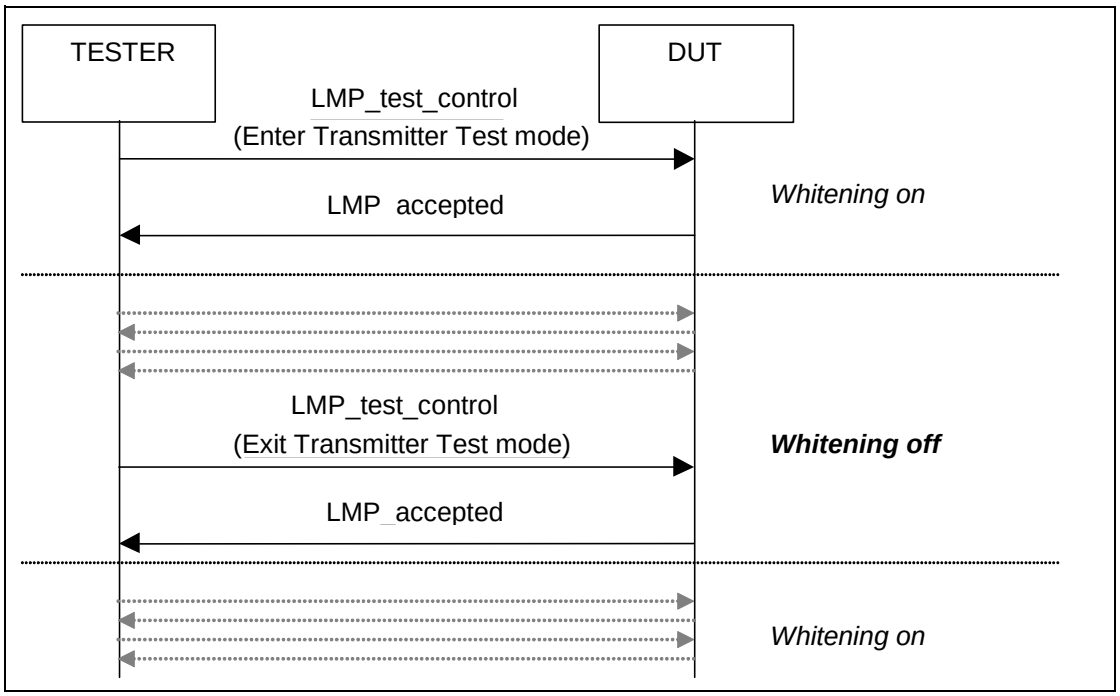



*Figure 1.4: Use of whitening in Transmitter mode*

### *1.1.2.2 Pseudorandom Sequence*

The same pseudorandom sequence of bits shall be used for each transmission (i.e. the packet is repeated). A PRBS-9 Sequence is used, see [2] and [3].

The properties of this sequence are as follows (see [3]). The sequence may be generated in a nine-stage shift register whose 5th and 9th stage outputs are added in a modulo-two addition stage (see Figure 1.5), and the result is fed back to the input of the first stage. The sequence begins with the first ONE of 9 consecutive ONEs; i.e. the shift register is initialized with nine ones.

- Number of shift register stages: 9
- Length of pseudo-random sequence: $2^9$-1 = 511 bits
- Longest sequence of zeros: 8 (non-inverted signal)

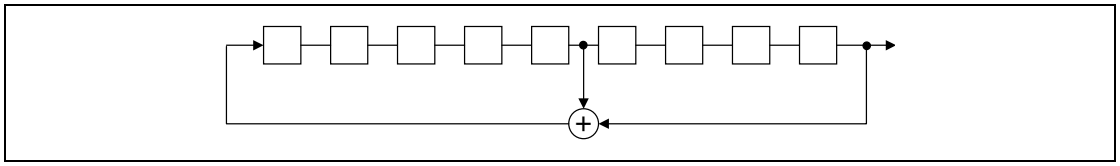


*Figure 1.5: Linear Feedback Shift Register for Generation of the PRBS sequence*

### *1.1.2.3 Control of Transmit Parameters*

The following parameters can be set to configure the transmitter test:

1. Bit pattern:
   - Constant zero
   - Constant one
   - Alternating 1010...[1]
   - Alternating 1111 0000 1111 0000...[4]
   - Pseudorandom bit pattern
   - Transmission off
2. Frequency selection:
   - Single frequency
   - Normal hopping
3. TX frequency
   - k $\Rightarrow$ f := (2402 + k) MHz
4. Default poll period in TDD frames (n * 1.25 ms)
5. Packet Type
6. Length of Test Sequence (user data of packet definition in “Baseband Specification” on page 59[vol. 2])

#### *1.1.2.4 Power Control*

When the legacy power control mechanism is tested the DUT shall start transmitting at the maximum power and shall reduce/increase its power by one step on every LMP_incr_power_req or LMP_decr_power_req PDU received. When the enhanced power control mechanism is tested a DUT shall start transmitting at the maximum power and shall reduce/increase its power by one step or go to the maximum power level when a LMP_power_control_req PDU is received.

#### *1.1.2.5 Switch Between Different Frequency Settings*

A change in the frequency selection becomes effective when the LMP procedure is completed:

When the tester receives the LMP_accepted it shall then transmit POLL packets containing the Ack for at least 8 slots (4 transmissions). When these transmissions have been completed the tester shall change to the new frequency hop and whitening settings.

After sending LMP_accepted the DUT shall wait for the LC level Ack for the LMP_accepted. When this is received it shall change to the new frequency hop and whitening settings.

There will be an implementation defined delay after sending the LMP_accepted before the TX or loopback test starts. Testers shall be able to cope with this.

1. It is recommended that the sequence starts with a one; but, as this is irrelevant for measurements, it is also allowed to start with a zero.

*Note: Loss of the LMP_accepted PDU will eventually lead to a loss of frequency synchronization that cannot be recovered. Similar problems occur in normal operation, when the hopping pattern changes.*

#### *1.1.2.6 Adaptive Frequency Hopping*

Adaptive Frequency Hopping (AFH) shall only be used when the Hopping Mode is set to 79 channels (e.g. Hopping Mode = 1) in the LMP_test_control PDU. If AFH is used, the normal LMP commands and procedures shall be used. When AFH is enabled prior to entering test mode it shall continue to be used with the same parameters if Hopping Mode = 1 until the AFH parameters are changed by the LMP_set_AFH PDU.

The channel classification reporting state shall be retained upon entering or exiting Test Mode. The DUT shall change the channel classification reporting state in Test Mode based on control messages from the tester (LMP_channel_classification_req) and from the Host (HCI Write_AFH_Channel_Classification_Mode).

### 1.1.3 LoopBack test

In loopback, the device under test receives normal baseband packets containing payload *Accepted* from the tester. The received packets shall be decoded in the DUT, and the payload shall be sent back using the same packet type. The return packet shall be sent back in either the slave-to-master transmission slot directly following the transmission of the tester, or it is delayed and sent back in the slave-to-master transmission slot after the next transmission of the tester (see Figure 1.7 to Figure 1.9 on page 419).

There is no signaling to determine or control the mode. The device behavior shall be fixed or adjusted by other means, and shall not change randomly.

The tester can select, whether whitening is on or off. This setting holds for both up- and downlink. For switching the whitening status, the same rules as in Section 1.1.2 on page 412 (Figure 1.4) shall apply.

The following rules apply (for illustration see Figure 1.6 on page 418):

- If the synch word was not detected, the DUT shall not reply.
- If the header error check (HEC) fails, the DUT shall either reply with a NULL packet with the ARQN bit set to NAK or send nothing.
- If the packet contains an LMP message relating to the control of the test mode this command shall be executed and the packet shall not be returned, though ACK or NAK shall be returned as per the usual procedure. Other LMP commands are ignored and no packet is returned.
- The payload FEC is decoded and the payload shall be encoded again for transmission. This allows testing of the FEC handling. If the pure bit error rate shall be determined the tester chooses a packet type without FEC.

- The CRC is shall be evaluated. In the case of a failure, ARQN=NAK shall be returned. The payload shall be returned as received.
  A new CRC for the return packet shall be calculated for the returned payload regardless of whether the CRC was valid or not.
- If the CRC fails for a packet with a CRC and a payload header, the number of bytes as indicated in the (possibly erroneous) payload header shall be looped back.

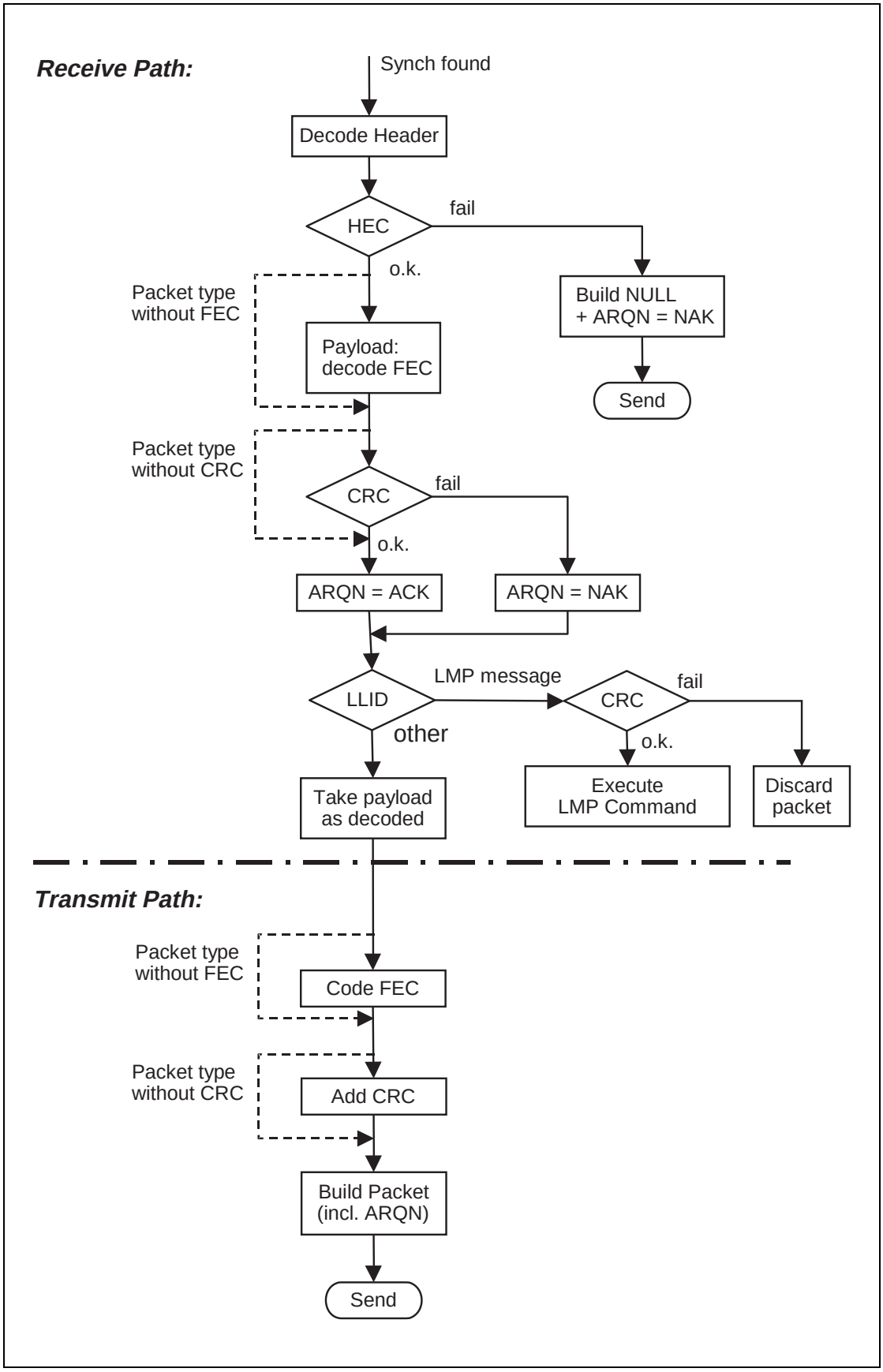



*Figure 1.6: DUT Packet Handling in Loop Back Test*

The timing for normal and delayed loopback is illustrated in Figure 1.7 to Figure 1.9:

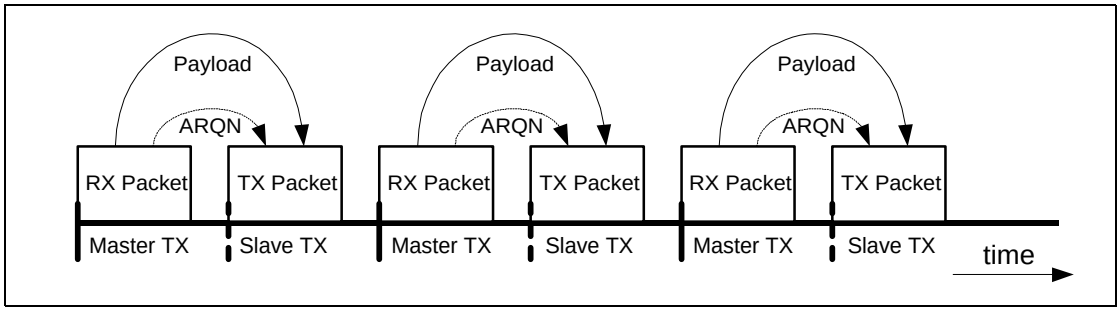



*Figure 1.7: Payload & ARQN handling in normal loopback.*

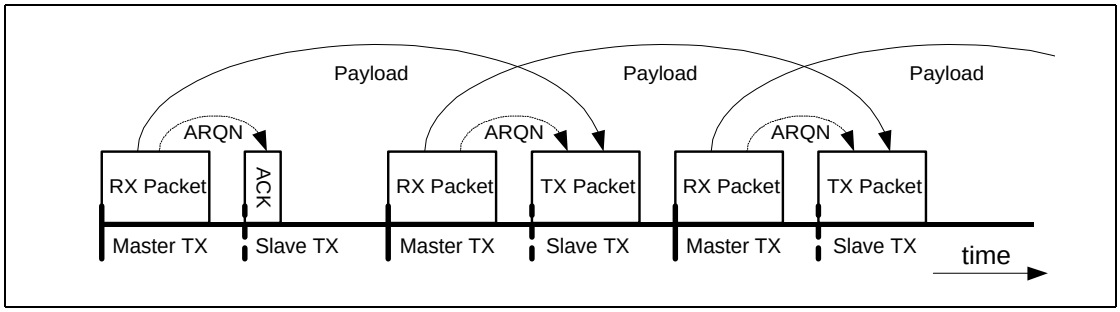



*Figure 1.8: Payload & ARQN handling in delayed loopback - start.*

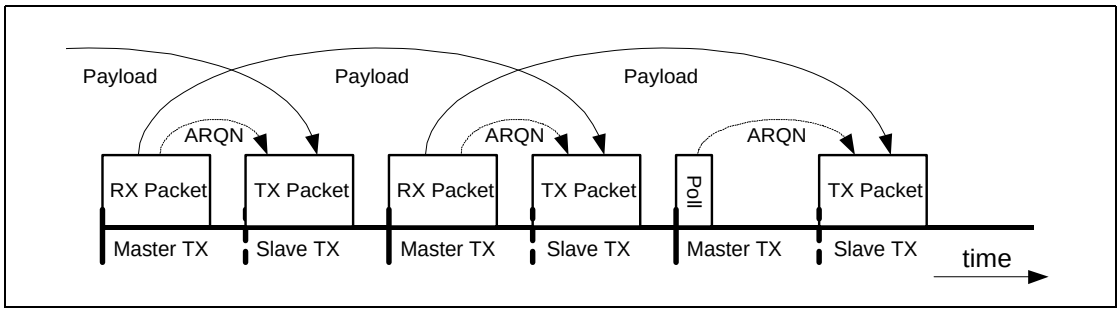



*Figure 1.9: Payload & ARQN handling in delayed loopback - end.*

The whitening is performed in the same way as it is used in normal active mode.

The following parameters can be set to configure the loop back test:

1. Packet Class[1]
   ACL Packets
   SCO Packets
   eSCO Packets
   ACL Packets without whitening

1. This is included because, in the future, the packet type numbering may not remain unambiguous.

SCO Packets without whitening
eSCO Packets without whitening

2. Frequency Selection
   Single frequency (independent for RX and TX)
   Normal hopping
3. Power level: (To be used according radio specification requirements)
   power control or fixed TX power

The switch of the frequency setting is done exactly as for the transmitter test (see Section 1.1.2.5 on page 415).

### 1.1.4 Pause test

Pause test is used by testers to put the device under test into Pause Test mode from either the loopback or transmitter test modes.

When an LMP_test_control PDU that specifies Pause Test is received the DUT shall stop the current test and enter Pause Test mode. In the case of a transmitter test this means that no more packets shall be transmitted. While in Pause Test mode the DUT shall respond normally to POLL packets (i.e. responds with a NULL packet). The DUT shall also respond normally to all the LMP packets that are allowed in test mode.

When the test scenario is set to Pause Test all the other fields in the LMP_test_control PDU shall be ignored. There shall be no change in hopping scheme or whitening as a result of a request to pause test.

## 1.2 AMP TEST SCENARIOS

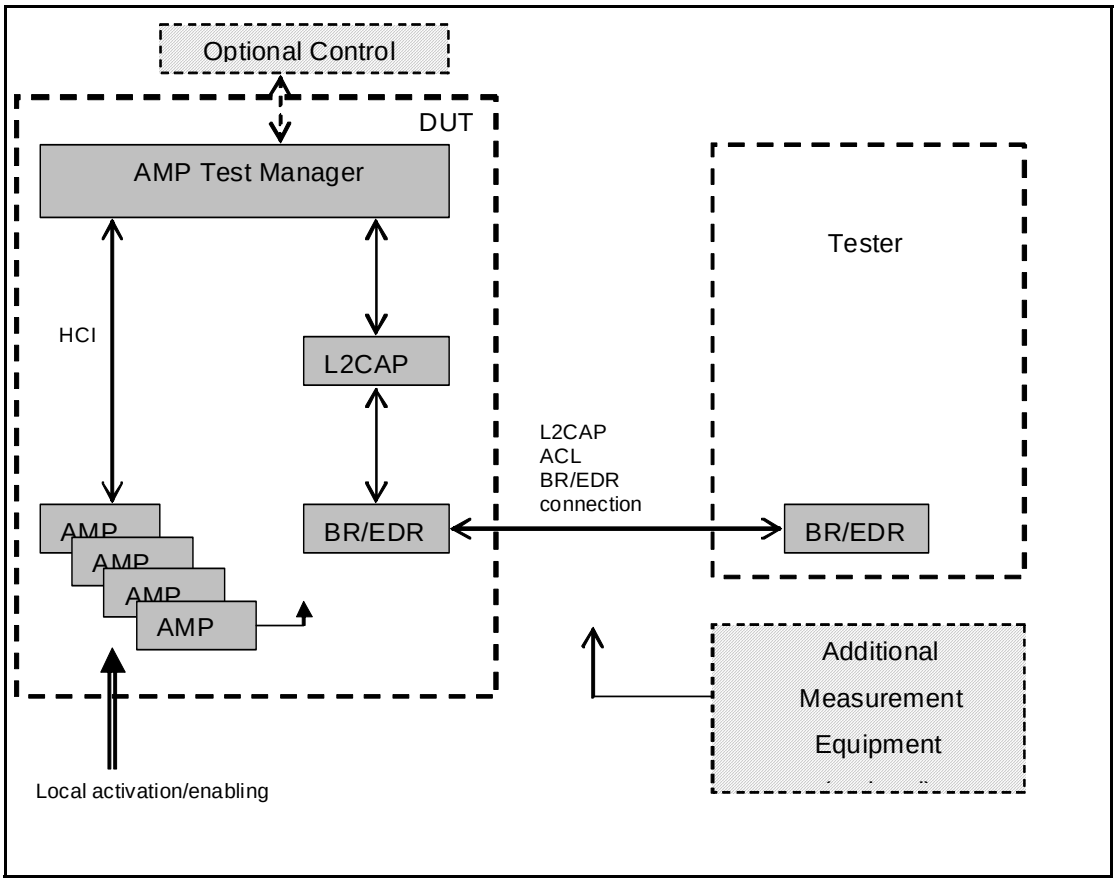



*Figure 1.10: Test Architecture*

### 1.2.1 Methodology Overview

To perform PHY tests on an AMP, the DUT shall first have AMP Test Mode enabled locally as required for a BR/EDR Controller using the HCI Enable_Device_Under_Test_Mode command. When this command is received by the AMP controller it shall enable the AMP to accept HCI Test Commands and when received by the BR/EDR controller it shall enable the AMP Test Manager to pass HCI commands and events between the tester and the AMP device utilizing the BR/EDR connection. A DUT AMP shall reject all AMP related HCI Test Commands, except HCI_Enable_Device_Under_Test_Mode, until the HCI_Enable_Device_Under_Test_Mode command has been issued by the local Host. Although AMP Test Mode is enabled the AMP shall not enter AMP Test Mode until the HCI_AMP_Test_Command is received.

Note: Enabling test mode with HCI_Enable_Device_Under_Test_Mode command allows the AMP to enter test mode when an HCI_AMP_Test_Command is received.

The local control of an AMP in AMP Test Mode shall be by the AMP Test Manager as shown in Figure 1.10. The AMP Test Manager will receive AMP Test Manager commands and HCI test commands via ACL data packets as shown

in Figure 1.10, using an L2CAP connectionless fixed test channel (channel 0x003F) from a Bluetooth test device. This does not exclude driving the AMP Test Manager directly from an implementation vendor specific interface as shown from the "Optional Control" in Figure 1.10.

The AMP Test Manager commands allow the tester to discover the number and type of AMPs supported and to read their PHY capability support via the AMP Test Manager.

The tester can use HCI commands and HCI Test Commands to control all of the AMPs to perform qualification, IOP testing as well as entering AMP Test Mode to perform AMP PHY tests. The HCI events and responses shall be routed back to the tester via the fixed test channel (0x003F).

If an HCI command is sent to an invalid Controller ID (a controller not reported in the Discovery Response) the AMP Command Rejected Event shall be returned with an Invalid Controller Id as the reason.

An AMP shall be taken out of AMP Test Mode by the HCI_Reset command. This will reset only the AMP receiving the HCI_Reset.

The AMP Test Manager shall not be enabled to support HCI commands over the fixed test channel (0x003F) unless AMP Test Mode has been enabled locally with the HCI_Enable_Device_Under_Test_Mode command.

The "Optional Control" interface shown in Figure 1.10 is a vendor specific interface that allows local control of the AMP Test Manager when a BR/EDR link may not be available.

#### *1.2.1.1 Initiation Example Description*

The AMP Test Manager is enabled to receive AMP test commands and HCI commands over the fixed test channel once it has been locally enabled. Using the fixed test channel, the tester shall use the AMP test commands and events to identify the number and type of AMPs available.

When the AMP Test Manager has been enabled and the AMPs identified the Tester shall use the AMP Test Manager to

- Transfer HCI commands and Events to and from the AMPs for qualification and IOP testing.
- Put the AMPs into test mode using HCI AMP test commands and events for AMP PHY testing.

### 1.2.2 Control and Configuration

Control and configuration of AMP tests shall be performed using two groups of commands/events over the ACL BR/EDR connection on the fixed test channel. The two groups are the AMP Test Manager commands/events which are to the

AMP Test Manager and the HCI AMP test commands/events which are routed via the AMP Test Manager between the AMP indicated as part of the message structure and the tester.

| | |
|---|---|
| AMP Test Manager commands and events | AMP Command Rejected Event |
| | AMP Discover Request Command |
| | AMP Discover Response Event |
| | AMP Read PHY Capability Bit Map Request |
| | AMP Read PHY Capability Bit Map Response Event |
| HCI AMP test commands and events | Read Loopback Mode Command |
| | Write Loopback Mode Command |
| | Enable AMP Receiver Reports Command |
| | AMP Test End Command |
| | AMP Test Command |
| | AMP Start Test Event |
| | AMP Test End Event |
| | AMP Receiver Report Event |

### 1.2.3 AMP Test Manager

The AMP Test Manager provides the interface to the AMPs available so that qualification, IOP and PHY testing can be performed using the BR/EDR connection. The AMP Test Manager provides the following services:

1. Identification of the number and type of AMPs available
2. Enumeration of the available AMPs so that they can be communicated to individually
3. Routing of the HCI commands and events between the AMPs and the tester over the ACL BR/EDR connection on the fixed test channel, or the vendor specific interface (that is, a local interface to the AMP Test Manager other than the BR/EDR radio).

### 1.2.4 Test Commands/Events Format

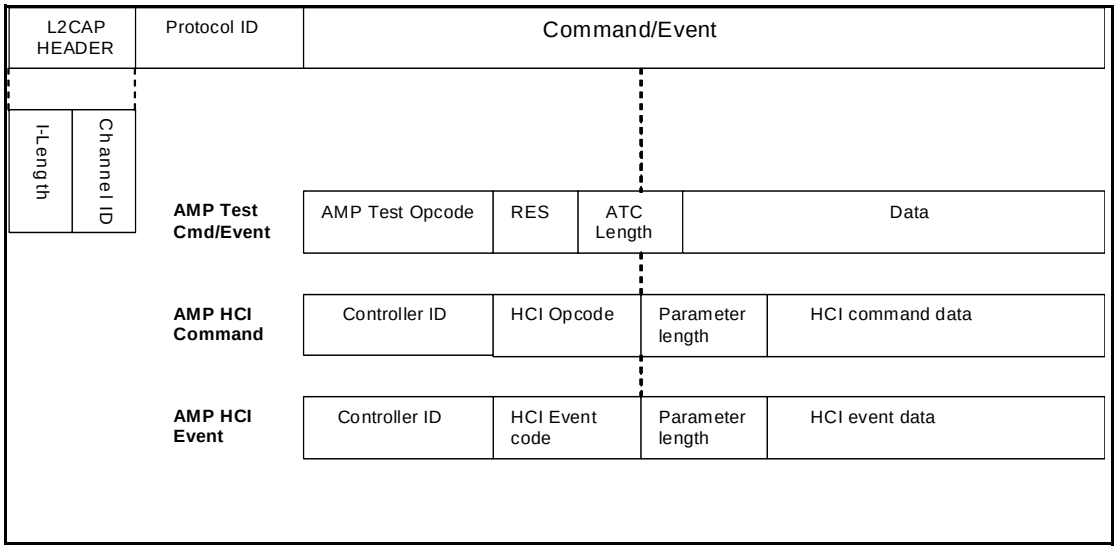



*Figure 1.11: L2CAP test command/event formats*

The fields shown are:

L2CAP header (4 octets)

This is the Basic L2CAP header with the Channel ID always set to test channel (0x003F). (See section [Vol. 3], Part A Section 3.1)

- *Length (2 octets)*

  Length of the Information payload which includes the Protocol ID and the Command/Event

- *Channel ID ( 2 octets)*

  Fixed value of (0x003F)

Protocol ID (1 octet)

This identifies the protocol in the command/event data. The IDs used are an extension of the UART transport packet types.

| Protocol ID | Command/event type |
|---|---|
| 0x01 | HCI Command |
| 0x02 | Reserved |
| 0x03 | Reserved |
| 0x04 | HCI Event |
| 0x05 | AMP Test Command |
| 0x06 | AMP Test Event |

*Table 1.1: L2CAP valid test protocol IDs*

Controller ID (1 octet)

Each Controller in a device is assigned a one-octet unique identifier by the local AMP Test Manager. This field determines the destination of the HCI Command or the source of the HCI Event. The Controller ID values shall be in the range of 1 to 255. Controller ID 0 has been reserved for the primary BR/EDR controller.

Note: "Controller" is a generic term which covers both the primary BR/EDR controller and any AMPs

ATO (AMP Test Opcode) (1 octet)

This is the AMP Test Manager commands or event code as described in Section 1.2.5.

RES (1 octet)

Reserved = 0

AMP Test Command (ATC) Length (2 octets)

This is the number of octets in the data portion of this message

AMP HCI Commands and Events

The format of these packets follow the standard HCI format described in [Vol. 2], Part E, after the Controller ID.

Examples

| | **Protocol ID** | **Command/Event** | | | |
|---|---|---|---|---|---|
| AMP Test command | 0x05 | AMP Test Opcode | 0x00 | Length | AMP Test Command |
| AMP Test Event | 0x06 | AMP Test Opcode | 0x00 | Length | AMP Test Event |
| AMP HCI Command | 0x01 | Controller ID | HCI Command | | |
| AMP HCI Event | 0x04 | Controller ID | HCI Event | | |

### 1.2.5 AMP Test Manager Commands/Events

The AMP Test Manager commands are sent to the AMP Test Manager from the tester and the events are from the AMP Test Manager to the tester.

The data fields are:

- *AMP Test Opcode (1 octet)*
  Identifies the type of AMP Test Manager command/event.
- *RES (1 octet)*
  Reserved and shall be set to 0.
- *Amp Test Command (ATC) length (2 octets)*
  Length of the data field of the command/event. This length does not include the AMP Test Opcode, reserved octet or the ATC length.

| AMP Test Opcode | Description |
|---|---|
| 0x00 | Reserved |
| 0x01 | AMP Command Rejected Event |
| 0x02 | AMP Discover Request |
| 0x03 | AMP Discover Response Event |
| 0x04 | AMP Read PHY Capability Bit Map Command |
| 0x05 | AMP Read PHY Capability Bit Map Response Event |
| 0x06 - 0xFF | Reserved |

*Table 1.2: AMP Test Opcodes*

### *1.2.5.1 AMP Command Rejected Event*

This event shall be sent by the DUT to the tester when a command cannot be accepted. The parameter returned indicates the reason.

| Opcode = 0x01 | Reserved = 0 | ATC length = 2 | Reason |
|---|---|---|---|

The data fields are:

- *AMP Test Opcode (1 octet)*
  0x01
- *RES (1 octet)*
  Reserved and shall be set to 0
- *ATC length (2 octets)*
  Always set to 0x0002
- *Reason (2 octets)*
  See Table 1.3.

.

| Reason Value | Description |
|---|---|
| 0x0000 | Command not recognized |
| 0x0001 | Invalid Controller ID |
| 0x0002 | Invalid Protocol ID |
| 0x0003 to 0xFFFF | Reserved |

*Table 1.3: AMP Command reasons for rejection*

### *1.2.5.2 AMP Discover Request*

The tester shall send this command to the AMP Test Manager to obtain the available Controllers and their types.

| Opcode = 0x02 | Reserved = 0 | ATC length = 0 |
|---|---|---|

The data fields are:

- *AMP Test Opcode (1 octet)*
  Shall be set to 0x02
- *RES (1 octet)*
  Reserved and shall be set to 0x00.
- *ATC length (2 octet)*
  Shall be set to 0x0000

Events generated

On receipt of the AMP_Discover_Request command the AMP Test Manager shall return the AMP_Discover_Response event.

### *1.2.5.3 AMP Discover Response Event*

When the AMP Test Manager receives an AMP_Discover_Request command it shall reply to the tester with an AMP_Discover_Response event indicating for each controller the Controller ID and Controller Type. The first Controller ID / Type pair in the response shall be 0 for the BR/EDR primary controller followed by the available additional Controllers ID and Type pairs.

| Opcode = 0x03 | Reseved = 0 | ATC length | Controller list |
|---|---|---|---|

**Controller list**

| Controller ID=0 | Controller Type=0 | Additional Controller ID and Type pairs |
|---|---|---|

- *AMP Test Opcode (1 octet)*

  Shall be set to 0x03
- *RES (1 octet)*

  Reserved and shall be set to 0x00.
- *ATC length (2 octets)*

  ATClength = (Controller ID length + Controller Type length) * number of Controllers
- *Controller Types (1 octet)*

  Each type of Controller is assigned a unique one-octet value. These values are defined in Assigned Numbers. A device may have multiple AMPs of the same type.

Note: The AMP Discovery Response shall include the primary BR/EDR Controller as the first Controller ID and Controller Type pair.

### *1.2.5.4 AMP Read PHY Capability Bit Map Command*

To perform AMP PHY tests the tester needs to know the supported PHY capabilities of the DUT. This information shall be provided in the PHY Capability Bit Map. This command shall be sent from the tester to the DUT to request the PHY Capability Bit Map from an AMP with the passed Controller ID.

| Opcode = 0x04 | Reserved = 0 | ATC length = 1 | Controller ID |
|---|---|---|---|

The data fields are:

- *AMP Test Opcode (1 octet)*

  Always set to 0x04
- *RES (1 octet)*

  Reserved and shall be set to 0
- *ATC length (2 octets)*

  Always set to 0x0001
- *Controller ID (1 octet)*

  The Controller from which the PHY Capability Bit Map is being requested. This is an invalid request from a BR/EDR Controller and will be rejected with the reason of Invalid Controller ID.

### *1.2.5.5 AMP Read PHY Capability Bit Map Response Event*

When the AMP_Read_PHY_Capability_Bit_Map Command is received by an AMP Test Manager it shall reply with the capability bit map for the AMP indicated if the Controller ID is valid.

- *AMP Test Opcode (1 octet)*

  Always set to 0x05
- *RES (1 octet)*

  Reserved and shall be set to 0
- *ATC length (2 octets)*

  The length of the PHY Capabilities Bit Map returned depending on the AMP type.
- *Status (1 octet)*

  See the following table.

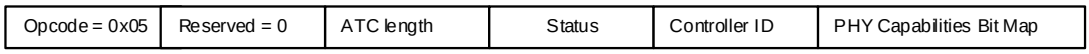
| Reason Value | Description |
|---|---|
| 0x00 | Success |
| 0x01 | Invalid Controller ID |
| 0x02 to 0xFF | Reserved |

Note: If the status is not Success the remaining parameters shall be zero.

- *Controller ID (1 octet)*

  This is the Controller ID from which the PHY Capability Bit Map has been sent.
- *PHY Capabilities Bit Map*

  See [Volume 5] for the PHY Capabilities Bit Map.




## 1.3 REFERENCES

[1] Bluetooth Link Manager Protocol.

[2] CCITT Recommendation O.153 (1992), Basic parameters for the measurement of error performance at bit rates below the primary rate.

[3] ITU-T Recommendation O.150 (1996), General requirements for instrumentation for performance measurements on digital transmission equipment.

[4] Bluetooth Baseband Specification.

# 2 TEST CONTROL INTERFACE (TCI)

This section describes the Bluetooth Test Control Interface (TCI). The TCI provides a uniform method of accessing the upper interface of the implementation being tested. This facilitates the use of a standardized interface on test equipment used for formal Qualification of implementations.

## 2.1 INTRODUCTION

### 2.1.1 Terms Used

| | |
|---|---|
| Conformance testing | Testing according to the applicable procedures given in the Bluetooth Protocol Test Specifications and the Bluetooth Profile Conformance Test Specification when tested against a test system. |
| HCI | Host Controller Interface |
| IUT | Implementation Under Test: An implementation of one or more Bluetooth protocols and profiles which is to be studied by testing. This term is used when describing the test concept for products and components equipped with Bluetooth wireless technology as defined in the PRD. |
| PRD | Bluetooth Qualification Program Reference Document: This document is maintained by the Bluetooth Qualification Review Board and is the reference to specify the functions, organization and processes inside the Bluetooth Qualification program. |
| TCI | Test Control Interface: The interface and protocol used by the test equipment to send and receive messages to and from the upper interface of the IUT. |

### 2.1.2 Usage of the Interface

For all products and components equipped with Bluetooth wireless technology, conformance testing is used to verify the implemented functionality in the lower layers. Conformance testing of the lowest layers requires an upper tester to test the implementation sufficiently well.

In order to avoid that the tester will have to adapt to each and every product and component equipped with Bluetooth wireless technology, the use of the standardized TCI is mandated. This concept puts some burden upon the manufacturer of the IUT in terms of supplying an adapter providing the necessary conversion from/to the IUT's specific interface to the TCI. The adapter can consist of hardware, firmware and software. After qualification testing has been performed the TCI may be removed from the product or component equipped with Bluetooth wireless technology. It is the manufacturer's option to remove it

from the qualified product or component equipped with Bluetooth wireless technology.

The TCI is used when qualifying the implemented functionality of the:

- Baseband layer, BB
- Link Manager layer, LM

If support of the Host Controller Interface is claimed by the manufacturer the TCI is used to qualify it.

## 2.2 TCI CONFIGURATIONS

This section describes the test configurations used when verifying the different Bluetooth requirements. Each layer in the Bluetooth stack is qualified using the procedures described in the layer specific test specification.

### 2.2.1 Bluetooth RF Requirements

For qualification of the Bluetooth Radio Frequency requirements the defined Test Mode is used, see Section 1 on page 411.

Similar to TCI, the specific test mode functionality may be removed from the product or component after qualification, at the discretion of the manufacturer.

#### *2.2.1.1 Required interfaces*

For RF qualification only the air interface is required, see Figure 2.1. Depending on the physical design of the IUT it might be necessary to temporarily attach an RF connector for executing the RF tests. As stated in Section 1 on page 411, the Test Mode shall be locally enabled on the IUT for security reasons. The implementation of this local enabling is not subject to standardization.

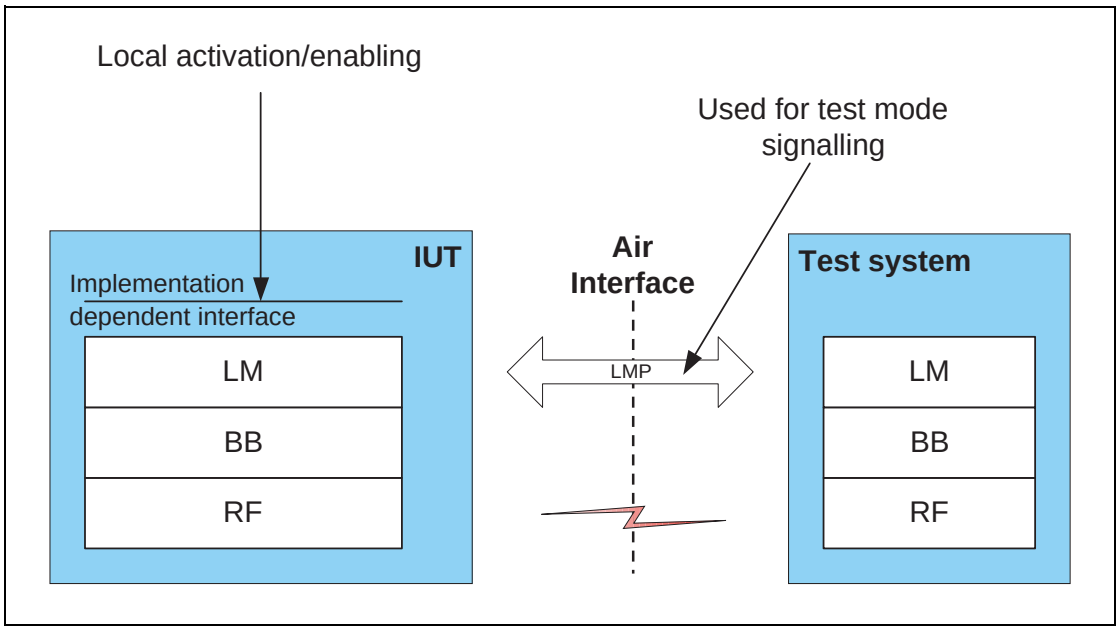



*Figure 2.1: General test set-up for RF qualification*

### 2.2.2 Bluetooth protocol requirements

Depending on which of the Bluetooth layers BB, LM or HCI are implemented in the product subject to qualification, the amount of testing needed to verify the Bluetooth protocol requirements differs. Also, the TCI used during the qualification is slightly different. The commands and events necessary for qualification are detailed in the test specifications and only those commands indicated in the test cases to be executed need be implemented.

For other protocols in the Bluetooth stack the TCI is not used. An implementation specific user interface is used to interact with the IUT's upper interface.

#### *2.2.2.1 Required interfaces*

For BB, LM and HCI qualification both the air interface of the IUT and the TCI are required. For other protocols both the air interface and the user interface are used as described in the test specification.

### 2.2.3 Bluetooth Profile Requirements

For each Bluetooth profile for which conformance is claimed, profile qualification testing is performed to verify the Bluetooth profile requirements. With higher layer protocols the TCI is not used during testing. A user interface specific to the implementation is used to interact with the IUT's upper interface.

#### *2.2.3.1 Required interfaces*

For this type of qualification both the air interface and the user interface of the IUT are used as described in the test specification.

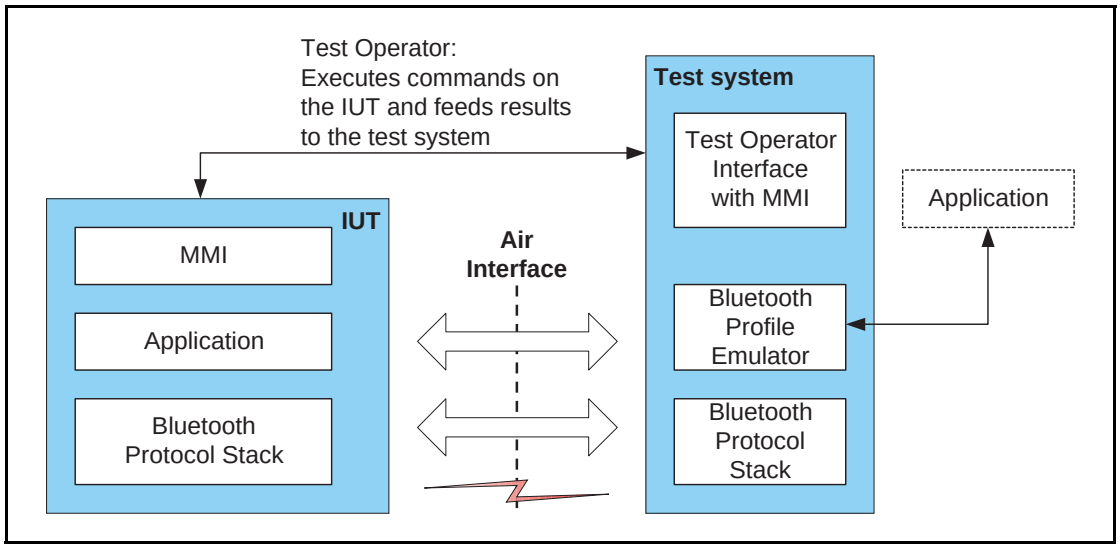



*Figure 2.2: General test set-up for profile qualification*

## 2.3 TCI CONFIGURATION AND USAGE

This interface is semantically and syntactically identical to the HCI interface described in “Host Controller Interface Functional Specification” on page 357[vol. 2]. The complete HCI interface is not required in the TCI, but the subset of HCI commands and events necessary for verifying the functionality of the IUT. The exact set of commands and events is specified in the test specifications for the layers subject to testing.

It is worth emphasizing again the TCI is an adapter, logically attached to the upper interface of the IUT. As such the TCI adapts the standardized signaling described here to the implementation specific interface of the IUT.

### 2.3.1 Transport Layers

The method used to convey commands and events between the tester and the IUT’s upper interface can be either physical bearer or “software bearer”.

#### *2.3.1.1 Physical bearer*

It is recommended to use one of the transport layers specified for the HCI in. However, other physical bearers are not excluded and may be provided on test equipment. The use of a physical bearer is required for test cases active in category A. Please see the PRD for the definitions of test case categories. Please see the current active Test Case Reference List for the categorization of specific test cases.

#### *2.3.1.2 Software bearer*

There is no physical connection between the tester and the IUT’s upper interface. In this case, the manufacturer of the IUT shall supply test software and hardware that can be operated by a test operator. The operator will receive instructions from the tester and will execute them on the IUT. The “software bearer” shall support the same functionality as if using the TCI with a physical bearer. Use of the “software bearer” shall be agreed upon between the manufacturer of the IUT and the test facility that performs the qualification tests. The test facilities can themselves specify requirements placed on such a “software bearer”. Furthermore, the use of a “software bearer” is restricted to test cases active in one of the three lower categories B, C and D.

### 2.3.2 Baseband and Link Manager Qualification

For the qualification of the link control part of the Baseband layer and for the Link Manager layer, the TCI is used as the interface between the test system and the upper interface of the IUT. The test system accesses the upper interface of the IUT by sending HCI commands and receiving HCI events from the IUT as described in the HCI specification. The required functionality on the TCI depends on the IUT's implemented functionality of the BB and LM layers, and therefore which test cases are executed.

A schematic example in Figure 2.3 shows the test configuration for BB and LM qualification of Bluetooth products which do not support HCI, and use a physical bearer for the TCI. In this example the Test Control (TC) Software represents what the manufacturer has to supply with the IUT when using an external test facility for qualification. The function of the TC Software is to adapt the implementation dependent interface to the TCI.

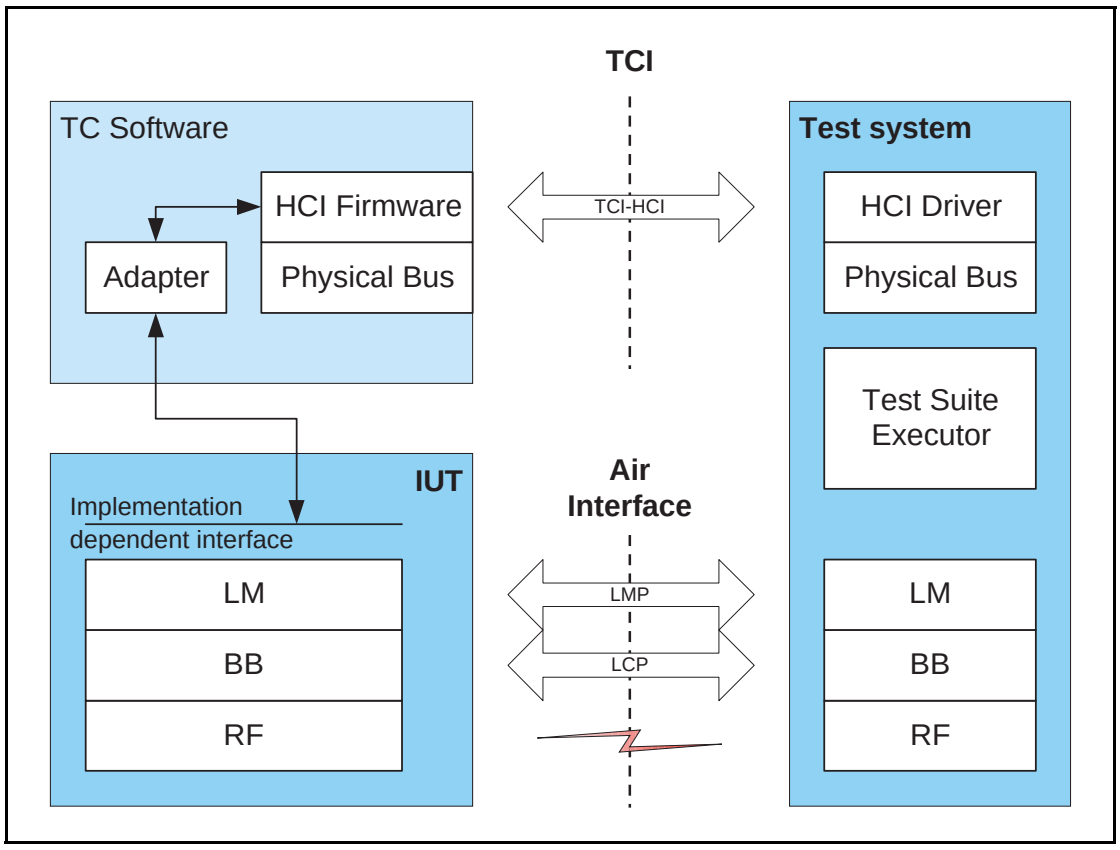



*Figure 2.3: BB and LM qualification with TCI physical bearer*

Figure 2.4 shows a schematic example of the test configuration for the same Bluetooth product using a "software bearer" for the TCI. Here the function of the Test Control Software is to represent the application that can be controlled by the test operator.

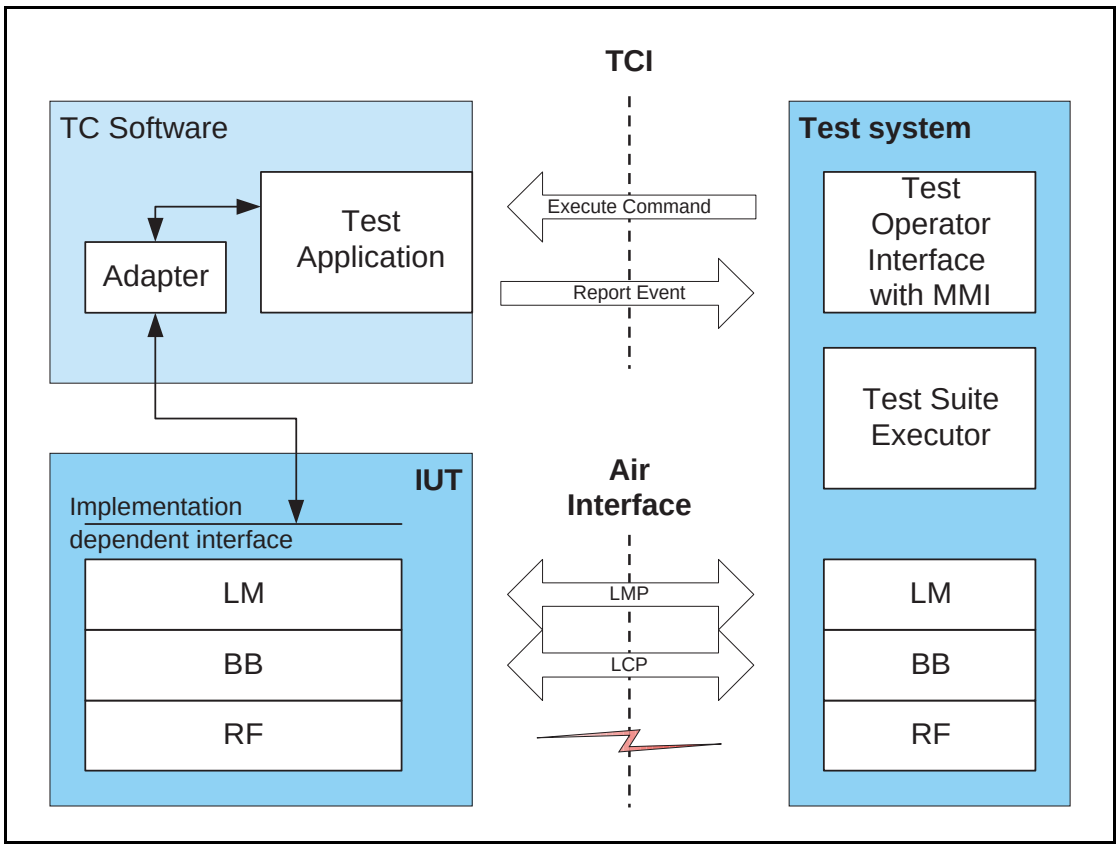



*Figure 2.4: BB and LM qualification with "software bearer"*

### 2.3.3 HCI Qualification

The TCI may also be used for HCI signaling verification and qualification. The HCI signaling is only verified if support of the HCI functionality is claimed by the manufacturer.

A schematic example in Figure 2.5 shows the test configuration for HCI qualification of Bluetooth products. As can be seen in the figure the implemented HCI is used as the interface to the tester.

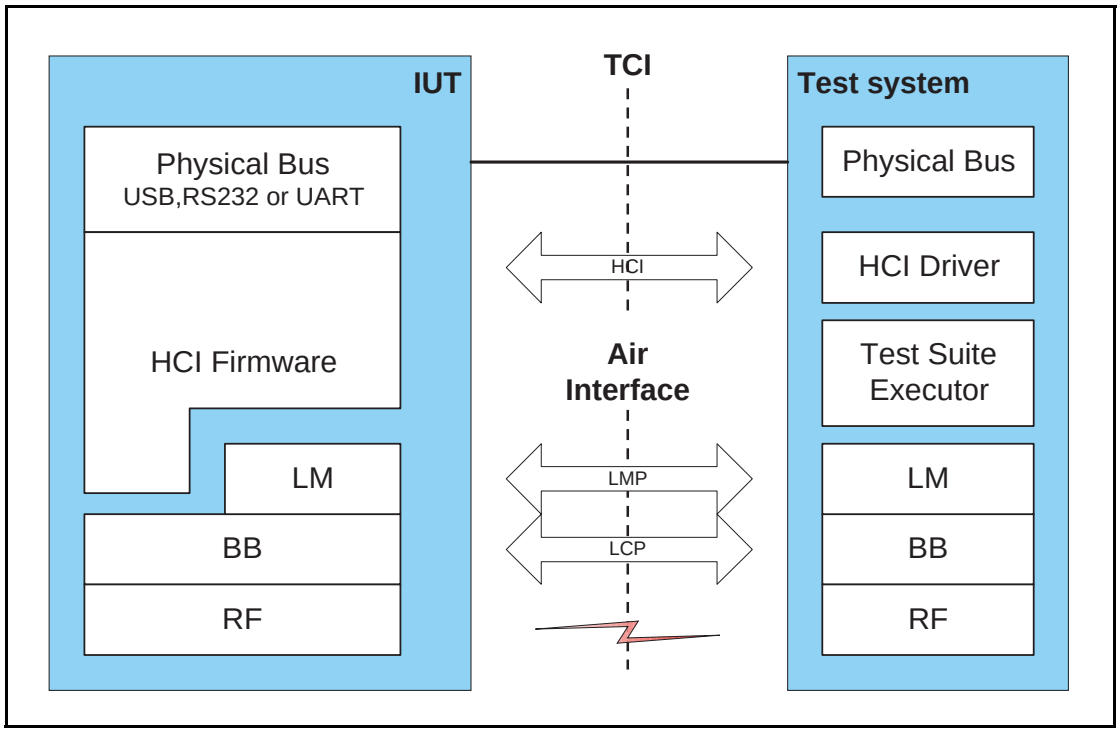



*Figure 2.5: General test set-up for HCI qualification*

# AMP MANAGER PROTOCOL SPECIFICATION

***This document specifies the changes to the Core specification required to incorporate the AMP Manager Protocol (A2MP) for the Alternate MAC/PHY feature.***


Ford Ex. 1052

# CONTENTS

**Contents** ..... **441**

**1 Introduction** ..... **442**

1.1 General Description ..... 442

**2 General Operation** ..... **443**

2.1 Basic Capabilities ..... 443

2.2 AMP Manager Channel Over L2CAP ..... 444

2.3 Using the AMP Manager Protocol ..... 445

2.3.1 Discovering a Remote AMP Manager ..... 445

2.3.2 Discovering Available Controllers on a Remote Device ..... 445

2.3.3 Creation of AMP Physical Links ..... 446

2.4 Controller IDs ..... 447

2.5 Controller Types ..... 447

**3 Protocol Description** ..... **448**

3.1 Packet Formats ..... 448

3.2 AMP Command Reject (Code 0x01) ..... 449

3.3 AMP Discover Request (Code 0x02) ..... 450

3.4 AMP Discover Response (Code 0x03) ..... 451

3.5 AMP Change Notify (Code 0x04) ..... 455

3.6 AMP Change Response (Code 0x05) ..... 456

3.7 AMP Get Info Request (Code 0x06) ..... 456

3.8 AMP Get Info Response (Code 0x07) ..... 456

3.9 AMP Get AMP Assoc Request (Code 0x08) ..... 458

3.10 AMP Get AMP Assoc Response (Code 0x09) ..... 459

3.11 AMP Create Physical Link Request (Code 0x0A) ..... 460

3.12 AMP Create Physical Link Response (Code 0x0B) ..... 461

3.13 AMP Disconnect Physical Link Request (Code 0x0C) ..... 462

3.14 AMP Disconnect Physical Link Response (Code 0x0D) ..... 463

3.15 Create Physical Link Collision Resolution ..... 464

3.16 Response Timeout ..... 465

3.17 Unexpected BR/EDR Physical Link Disconnect ..... 465

**Figures** ..... **466**

**Tables** ..... **467**

# 1 INTRODUCTION

## 1.1 GENERAL DESCRIPTION

The AMP Manager Protocol (A2MP) provides a means for one device to solicit information regarding AMP capabilities from another device. Each device contains an abstract entity called an AMP Manager which uses the AMP Manager Protocol to communicate with a peer AMP Manager on another device.

# 2 GENERAL OPERATION

## 2.1 BASIC CAPABILITIES

The AMP Manager entity has the following responsibilities and capabilities:

1. Ability to discover remote AMP Managers.
2. Ability to discover available Controllers
3. Ability to query remote AMP Controller information
4. Ability to manage AMP physical links.
5. Responsible for the creation of dedicated AMP keys

The AMP Manager communicates with the local AMP PAL and communicates with remote AMP Managers using the AMP Manager Protocol over a fixed L2CAP channel. The AMP Manager entity exists within the Bluetooth stack as depicted in Figure 2.1.

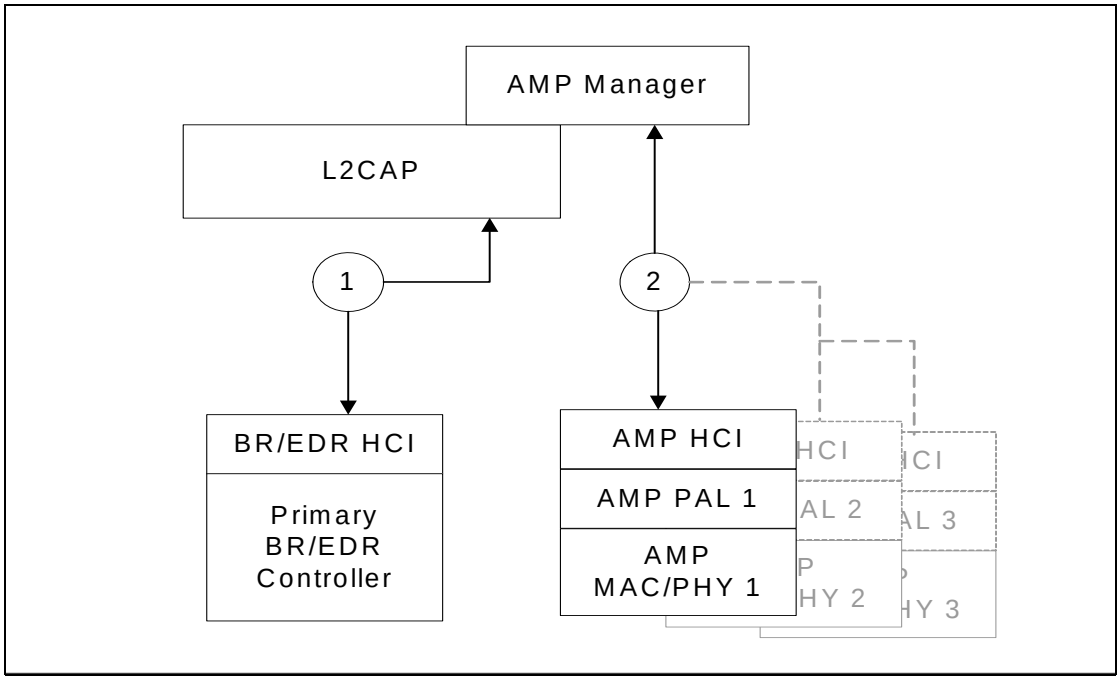



*Figure 2.1: Overview of the Lower Software Layers*

In original Figure 3.1, between steps 9 and 10 optional data flowed. Here we propose that a host can inquire on an AMP Controller's capabilities and eventually perform a move channel over to that AMP having created a Physical and Logical Links.

Figure 2.1 shows the two basic communication paths for the AMP Manager. The AMP Manager uses path 1 via L2CAP and the Primary BR/EDR Controller to first discover remote AMP Managers. The AMP Manager can query for the possibility of available remote AMPs and their capabilities as well as other

information. The AMP Managers communicate over the AMP Manager Protocol Channel (A2MP Channel). See Section 2.2 for more information on the A2MP Channel.

After discovering the AMPs supported by the remote device, the AMP Manager uses communication path 2 to manage the physical links between its local AMP(s) and AMPs on remote devices. Note that there can be more than one local AMP and there also may be more than one AMP located on the remote device.

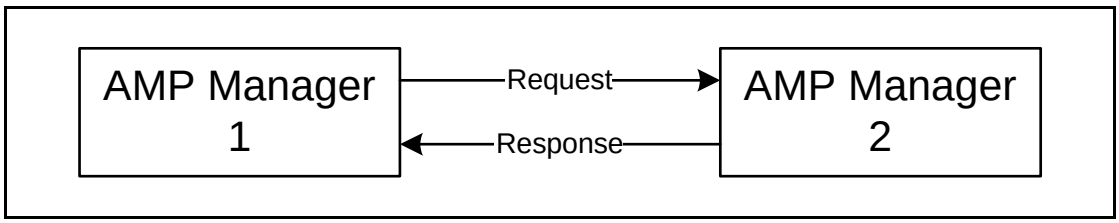



*Figure 2.2: AMP Manager Peers*

The AMP Manager Protocol is a request / response protocol between two AMP Managers. Requests and responses go in both directions. Either peer can request the same category of information from its corresponding peer.

## 2.2 AMP MANAGER CHANNEL OVER L2CAP

Peer AMP Managers communicate over the A2MP Channel which is an L2CAP fixed channel. The configuration parameters for the AMP Manager channel shall be as shown below. The MTU value shall be any valid value greater than or equal to 670 bytes.

| Parameter | Value |
|---|---|
| MTU | ≥ 670 |
| Flush Timeout | 0xFFFF (Infinite) |
| QoS | Best Effort |
| Mode | Enhanced Retransmission Mode |
| FCS Option | 16-bit FCS |

*Table 2.1: AMP Manager Channel configuration parameters*

The parameters for the Enhanced Retransmission Mode used on the fixed channel shall be as shown below. The Retransmission Time-out, Monitor Time-out and MPS shall be any valid value that is greater than or equal to the values stated in the following tables.

The minimum value for both MTU and MPS is 670 bytes. A larger MTU and MPS size can be communicated using the AMP Discover request (see section 3.3) and AMP Discover response (see Section 3.4) packets. The MTU parame-

| Parameter | Value |
|---|---|
| TxWindow size | 1 |
| Max Transmit | 0xff |
| Retransmission time-out | ≥ 2 seconds |
| Monitor time-out | ≥ 12 seconds |
| Maximum PDU size (MPS) | ≥ 670 |
| 16-bit FCS | Enabled |

*Table 2.2: AMP Manager Channel Enhanced Retransmission Mode parameters*

ter and the Enhanced Retransmission mode MPS parameter are numerically equal to allow an AMP manager to send any A2MP packet using a single L2CAP PDU. A device may segment any A2MP packet into multiple L2CAP PDUs.

## 2.3 USING THE AMP MANAGER PROTOCOL

The following text describes how the AMP Manager Protocol is used to establish an AMP physical link between two devices.

### 2.3.1 Discovering a Remote AMP Manager

Once an ACL connection is established over the Primary BR/EDR Controller, the local AMP Manager shall examine the L2CAP extended features bits of the remote device to determine if L2CAP fixed channels are supported (see [Part A], Logical Link Control and Adaptation Protocol Specification, Section 4.12 on page 69). If so, a query for the existence of its peer (AMP Manager) may be made by issuing an Information Request for all available fixed channels.

### 2.3.2 Discovering Available Controllers on a Remote Device

The next step is to discover the existence and capabilities of the AMP Controllers of the peer. An AMP Discover Request packet can be issued over the AMP Manager Protocol channel for this purpose.

In an AMP Discover Response packet, the peer AMP Manager will return information about its local AMP capabilities in the form of a Controller List. A Controller List is a sequence of Controller ID/Controller Type/Controller Status triples identifying the Controllers that are available for communication.

The Controller List contains a basic top level description of the available peer Controllers and their status. The AMP Manager parses this list for physical radio compatibility. If more information is required about a remote AMP Controller, the local AMP Manager can subsequently issue an AMP Get Info Request packet using the Controller ID retrieved via the AMP Discover exchange. An AMP Get Info Response packet is returned from the peer with AMP Controller

Info for the specified AMP Controller. The AMP Controller Info contains information about the AMP Controller that can be used by the host to determine whether or not to create a physical link to the AMP Controller.

### 2.3.3 Creation of AMP Physical Links

Once it has been determined that a Bluetooth connection over an AMP is desired, the AMP Manager retrieves the remote AMP Controller's AMP_Assoc structure by issuing an AMP Get AMP Assoc Request packet. It then passes the AMP_Assoc structure received from the remote device to its local AMP PAL and signals it to begin the physical discovery and connection process. In devices that support HCI this is done via the HCI_Create_Physical_Link command followed by the HCI_Write_Remote_AMP_ASSOC command.

Note that the AMP Manager only holds the contents of the AMP_Assoc structure for use in AMP Manager operations. The AMP Manager does not directly attempt to interpret the contents of the AMP_Assoc structure.

The AMP manager is responsible for generating the Physical Link Handle used in the HCI_Create_Physical_Link and HCI_Accept_Physical_Link commands. See section Host Controller Interface Functional Specification, Section 5.3.2, on page 426 for details. When the AMP PAL is ready it will generate an HCI_Channel_Selected event which is a signal to read the local AMP_Assoc structure using the HCI_Read_Local_AMP_ASSOC command.

The AMP Manager shall not create or accept a physical link over an AMP if mutual authentication was not performed or encryption was not enabled over BR/EDR.

When the AMP Manager does not have a dedicated AMP key for the selected Controller type between the two devices and the BR/EDR link key type is not "Debug", the AMP Manager shall call the Secure Simple Pairing h2 function (see [Vol 2 Part H] Section 7.7.5.2 on page 1107 to generate a dedicated AMP key. This dedicated AMP key shall be used during subsequent connections.

If the Bluetooth device supports more than one AMP type, the AMP Manager shall call the h2 function a second time to regenerate Key_GAMP as defined in [Vol 2 Part H] Section 7.7.5.3 on page 1107 before generating a Dedicated AMP link key for a different AMP type.

When the BR/EDR link key type is set to "Debug", the AMP Manager shall use Key_GAMP directly as the key for the AMP regardless of the AMP Type. The key length, key type and the link key itself are passed to the AMP Controller in the HCI_Create_Physical_Link command.

After receiving the HCI_Channel_Selected event and reading the local AMP_Assoc structure the AMP Manager then issues an AMP Create Physical Link Request packet to send the local AMP_Assoc structure to the peer device in order to engage the peer device in the AMP physical link creation.

If the peer accepts the request it issues an HCI_Accept_Physical_Link command to its local AMP and passes the AMP_Assoc structure received in the AMP Create Physical Link Request packet to its local AMP Controller. The peer responds with an AMP Create Physical Link Response packet to either accept or reject the request to create a physical link.

Both hosts will receive an HCI_Physical_Link_Complete event indicating the completion of the physical link creation.

On success, the procedure for creation of a logical connection can then commence.

## 2.4 CONTROLLER IDS

Each Controller in a device is assigned a one-octet unique identifier by the local AMP Manager. From the perspective of a remote device, Controller IDs are only valid during the life time of the AMP Manager Protocol channel. The Controller IDs on a remote device become invalid when the A2MP Channel is disconnected. If a local Controller becomes unavailable during the life time of the A2MP channel, the Controller ID may be reclaimed after receiving the AMP Change Response to the AMP Change Notify packets sent to all affected remote AMP Managers. It can then be assigned to any new AMP that is added. The value 0 is reserved for the Primary BR/EDR Controller. Other Controllers in the device shall be assigned values from 1 to 255.

If an AMP Controller becomes unavailable and later becomes available again during the life time of the A2MP Channel, remote AMP Managers shall not assume that the AMP Controller will be assigned the Controller ID that it was previously assigned.

## 2.5 CONTROLLER TYPES

Each type of Controller is assigned a unique one-octet value. These values are defined in Assigned Numbers. A device may have multiple AMPs of the same type.

# 3 PROTOCOL DESCRIPTION

The AMP Manager Protocol defines the procedures and format of the packets used to exchange information between two peer AMP Managers.

## 3.1 PACKET FORMATS

This section describes the packets used in the AMP Manager Protocol. Unless otherwise stated an implementation shall include all specified fields in a packet.

Figure 3.1 displays the general format of all AMP Manager Protocol packets.

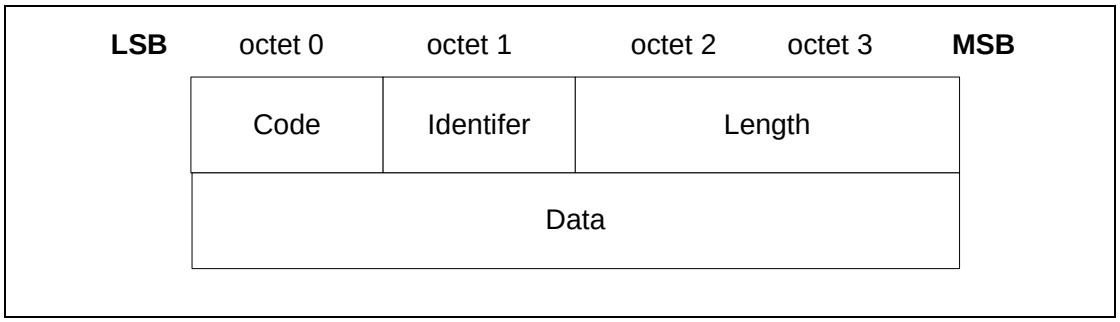



*Figure 3.1: AMP Manager Protocol Packet Format*

The fields shown are:

- *Code (1 octet)*

  The code identifies the type of packet. When an AMP manager receives a packet with an unknown Code field it shall send a Command Reject packet in response. Table 3.1 lists the codes defined by this document. All codes are specified with the most significant bit in the left most position.

| Code | Description |
|---|---|
| 0x00 | Reserved |
| 0x01 | AMP Command Reject |
| 0x02 | AMP Discover request |
| 0x03 | AMP Discover response |
| 0x04 | AMP Change notify |
| 0x05 | AMP Change response |
| 0x06 | AMP Get Info request |
| 0x07 | AMP Get Info response |
| 0x08 | AMP Get AMP Assoc request |

*Table 3.1: Codes*

| Code | Description |
|---|---|
| 0x09 | AMP Get AMP Assoc response |
| 0x0A | AMP Create Physical Link request |
| 0x0B | AMP Create Physical Link response |
| 0x0C | AMP Disconnect Physical Link request |
| 0x0D | AMP Disconnect Physical Link response |
| 0x0E - 0xFF | Reserved |

*Table 3.1: Codes*

- *Identifier (1 octet)*

  The Identifier field matches responses with requests. The requesting device sets this field and the responding devices uses the same value in the response. For each A2MP channel an AMP manger shall use different identifiers for each request sent (this includes the AMP Change Notify command). Following the original transmission of an identifier in a request, the identifier may be recycled if all other identifiers have subsequently been used.

  A response packet with an invalid identifier is discarded. Identifier 0x00 is an illegal identifier and shall never be used in any packet. Table 3.2 shows the valid values for the Identifier field,

| Value | Description |
|---|---|
| 0x00 | Invalid |
| 0x01 - 0xFF | Valid |

*Table 3.2: Identifier*

- *Length (2 octets)*

  The Length field indicates the size in octets of the data field of the packet only, i.e. it does not cover the Code, Identifier and Length fields.

- *Data (0 or more octets)*

  The Data field is variable in length. The Code field determines the format of the Data field. The Length field determines the length of the Data field.

## 3.2 AMP COMMAND REJECT (CODE 0X01)

An AMP Manager shall send an AMP Command Reject packet in response to a request packet with an unknown code or when sending the corresponding response is inappropriate. The identifier shall match the identifier of the request packet being rejected. AMP Command Reject packets shall not be sent in response to an unknown response packet.

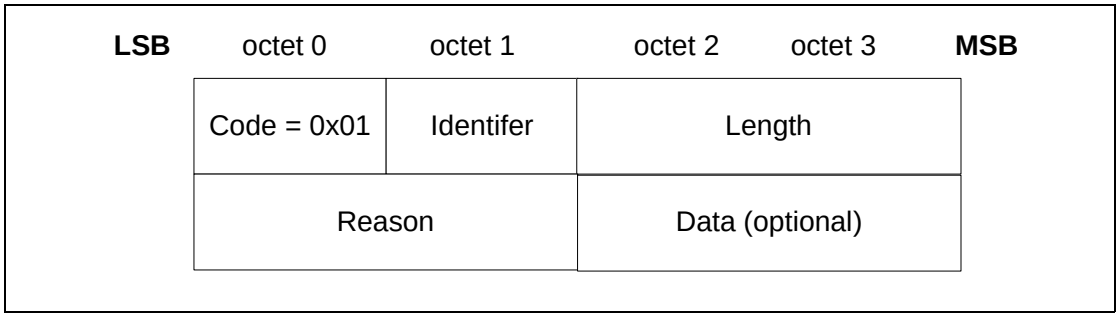



*Figure 3.2: AMP Command Reject Packet*

The data fields are

- *Reason (2 octets)*

  The Reason field describes why the request packet was rejected, and is set to one of the following Reason codes.

| Reason value | Description |
|---|---|
| 0x0000 | Command not recognized |
| Other | Reserved |

*Table 3.3: AMP Command Reject reasons*

- *Data (0 or more octets)*

  The length and content of the Data field depends on the Reason code. If the reason code is 0x0000 "Command not recognized" no Data field is used.

## 3.3 AMP DISCOVER REQUEST (CODE 0X02)

An AMP Manager sends an AMP Discover Request packet to a peer AMP Manager to obtain the list of the available Controllers supported by the peer device.

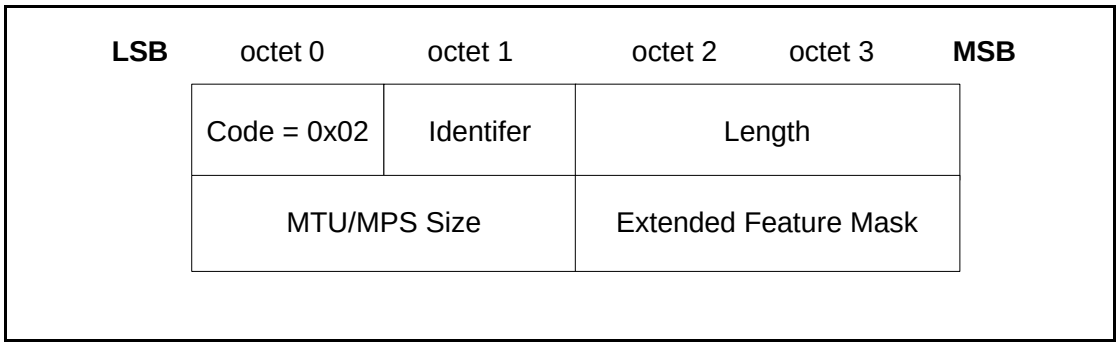



*Figure 3.3: AMP Discover Request Packet*

The data fields are

- *MTU/MPS size (2 octets)*

The MTU/MPS Size field in the AMP Discover Request packet indicates the MTU/MPS that the sender of the request can receive on the AMP Manager Protocol Channel. The minimum value is 670 (see Section 2.2). An AMP manager shall send the same value for the AMP Manager Protocol Channel MTU/MPS in both AMP Discover Request and AMP Discover Response packets.

- *Extended Feature Mask (2 octets)*

  The extended features supported by the AMP Manager are represented by a bit mask. Each feature is represented by a single bit which shall be set to 1 if the feature is supported and set to 0 otherwise. All reserved feature bits shall be set to 0 by the sender.

  The feature mask is defined in Table 3.4.

| Supported Feature | Octet | Bit |
|---|---|---|
| Reserved | 0 | 0-7 |
| Reserved | 1 | 0-6 |
| Extension Bit | 1 | 7 |

*Table 3.4: A2MP Extended Feature Mask*

Bit 7 of Octet 1 of this field is used to extend the A2MP Extended Feature Mask. If the Extension Bit is set then the Extended Feature Mask field is extended by two octets. Each two octet extension includes an Extension Bit that allows a further two octets to be added to the field. Figure 3.4 shows how the Extension Bit (E) can be used to extend the A2MP Extended Feature Mask.

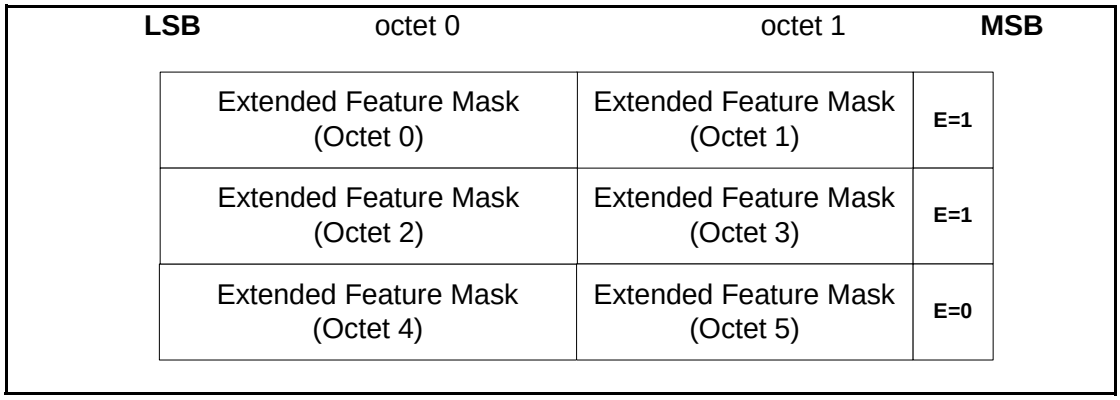



*Figure 3.4: Example A2MP Extended Feature Mask*

## 3.4 AMP DISCOVER RESPONSE (CODE 0X03)

When an AMP Manager receives an AMP Discover Request packet it shall reply with an AMP Discover Response packet indicating the IDs, types and status of the controllers that are currently available on the local device. If there is a change in the number or status of supported controllers after sending an AMP

Discover Response packet and before the A2MP Channel is disconnected, the AMP Manager shall send an AMP Change Notify packet to the peer AMP Manager.

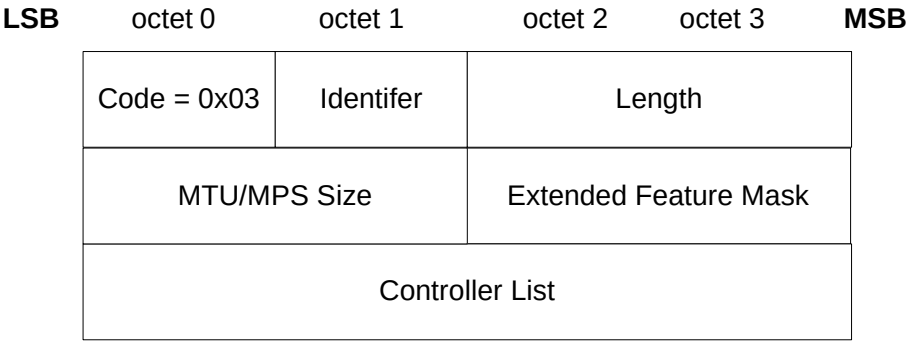



*Figure 3.5: AMP Discover Response Packet*

The data fields are

- *MTU/MPS size (2 octets)*

  The MTU/MPS Size field in the AMP Discover Response packet indicates the MTU/MPS that the sender of the response can receive on the AMP Manager Protocol Channel. The minimum value is 670 (see Section 2.2 ). An AMP Manager shall send the same value for the AMP Manager Protocol Channel MTU/MPS supported in both AMP Discover Request and AMP Discover Response packets.

- *Extended Feature Mask (2 octets)*

  The extended features supported by the AMP Manager are represented by a bit mask. Each feature is represented by a single bit which shall be set to 1 if the feature is supported and set to 0 otherwise. All unknown, reserved, or unassigned feature bits shall be set to 0.

  The feature mask is defined in Table 3.4 and Figure 3.4.

- *Controller List (3 or more octets)*

  Controller List is variable in length. It consists of 1 or more entry. Each entry is comprised of a Controller ID, a Controller Type and a Controller status.

  An entry for Controller ID 0x00 (Primary BR/EDR controller) shall always be sent and shall be the first entry in the Controller List. The Controller Type for this entry shall be set to 0x00 (Primary BR/EDR controller). The Controller status for this entry shall be set to 0x01 (The AMP Controller can only be used by Bluetooth technology). If the Primary BR/EDR Controller's Link Manager does not support Secure Simple Pairing, it shall be the only Controller in the Controller List.

  Figure 3.6 displays the format of the Controller list.

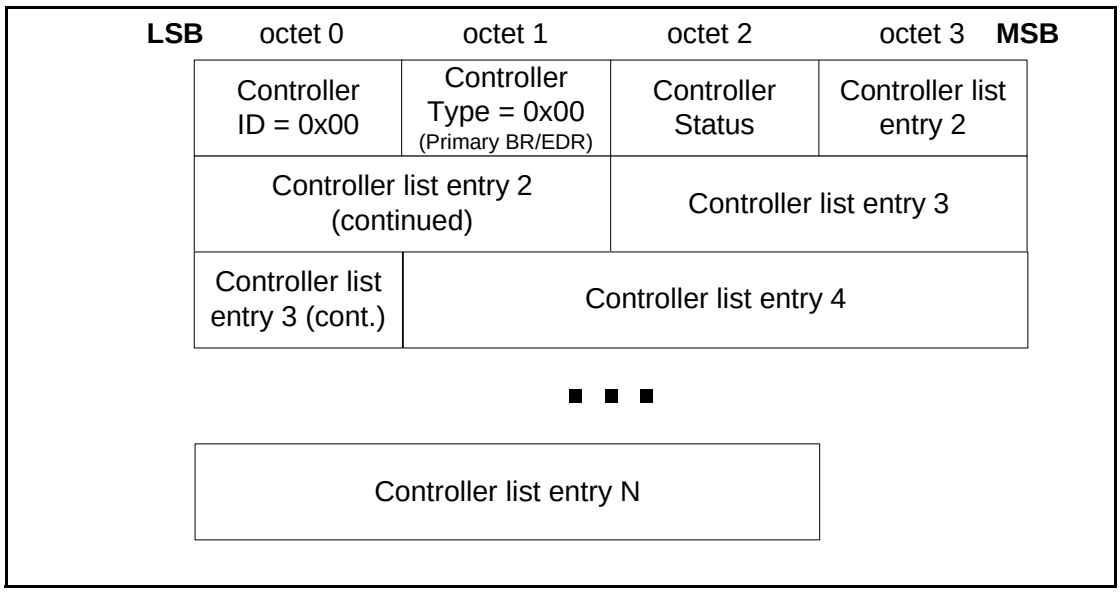



*Figure 3.6: Controller list format*

- *Controller ID (1 octet)*

  The Controller ID uniquely identifies a Controller (see Section 2.4 for more information about Controller IDs).

- *Controller Type (1 octet)*

  The Controller Type specifies the type of the Controller. The Controller type values are defined in Assigned Numbers.

- *Controller Status (1 octet)*

  The Controller Status field indicates the current status of the AMP. The status information can be used by the receiving device to help decide if it should use the AMP. If multiple AMPs are available the status can also be used by an implementation to determine which would be the optimal AMP to use. None of the AMP status values indicate that a Physical Link shall not be created to the AMP. If an implementation wishes to ensure it does not receive Create Physical Link Requests for a given AMP then it should remove the associated entry from the Controller list it sends.

Table 3.5 contains the different values of the Controller Status field.

| Value | Parameter Description |
|---|---|
| 0x00 | The Controller radio is available but is currently physically powered down.<br>This value should be used if the AMP Controller is present and can be powered up by the AMP Manager. A Controller List entry shall not exist for an AMP that is not present or cannot be powered up by the AMP manager.<br>This value indicates that there may be a cost of time and power to use this AMP Controller (i.e., the time taken and power required to power up the AMP Controller). These costs are AMP type and AMP implementation dependent. |

| Value | Parameter Description |
|---|---|
| 0x01 | This value indicates that the AMP Controller is only used by the Bluetooth technology and will not be shared with other non-Bluetooth technologies.<br>This value shall only be used if the AMP Controller is powered up.<br>This value does not indicate how much bandwidth is currently free on the AMP Controller. |
| 0x02 | The AMP Controller has no capacity available for Bluetooth operation.<br>This value indicates that all of the AMP Controller's bandwidth is currently allocated to servicing a non Bluetooth technology.<br>A device may create a Physical Link to an AMP Controller that has this status.<br>This value shall only be used if the AMP Controller is powered up. |
| 0x03 | The AMP Controller has low capacity available for Bluetooth operation.<br>This value indicates that the majority of the AMP Controller's bandwidth is currently allocated to servicing a non Bluetooth technology.<br>An AMP Controller with capacity in the approximate range of 0 < capacity < 30% should indicate this value.<br>This value does not indicate how much of the capacity available for Bluetooth operation is currently being used..<br>This value shall only be used if the AMP Controller is powered up. |
| 0x04 | The AMP Controller has medium capacity available for Bluetooth operation.<br>An AMP Controller with capacity in the approximate range of 30% < capacity < 70% should indicate this value.<br>This value does not indicate how much of the capacity available for Bluetooth operation is currently being used.<br>This value shall only be used if the AMP Controller is powered up. |
| 0x05 | The AMP Controller has high capacity available for Bluetooth operation<br>This value indicates that the majority of the AMP Controller's bandwidth is currently allocated to servicing the Bluetooth technology.<br>An AMP Controller with capacity in the approximate range of 70% < capacity < 100% should indicate this value..<br>This value does not indicate how much of the capacity available for Bluetooth operation is currently being used.<br>This value shall only be used if the AMP Controller is powered up. |
| 0x06 | The AMP Controller has full capacity available for Bluetooth operation<br>This value indicates that while currently the AMP is only being used by Bluetooth the device allows a different technology to share the radio.<br>This value shall be used as the default Controller Status value for devices that are not capable of determining the available capacity for Bluetooth operation for an AMP Controller.<br>This value does not indicate how much of the capacity available for Bluetooth operation is currently being used.<br>This value shall only be used if the AMP Controller is powered up. |

| Value | Parameter Description |
|---|---|
| Other | Reserved |

*Table 3.5: Controller Status*

The available capacity ranges given for values 0x03, 0x04, and 0x05 are only approximate and permit an implementation to not send an AMP Change Notify packet if the current availability is moving frequently from one range to another (e.g., if the current availability is moving between 68% (corresponding to Controller status 0x04) and 72% (corresponding to Controller status 0x05) frequently an implementation is not mandated to send an AMP Change Notify packet every time the value crosses the approximate 70% threshold between the two Controller Status values).

## 3.5 AMP CHANGE NOTIFY (CODE 0X04)

The AMP Change Notify packet indicates to the peer device that something has changed in the list of supported Controllers. An AMP Change Notify packet shall only be sent to a remote AMP Manager if it has previously requested a Controller List via an AMP Discover Request. Examples of when an AMP Change Notify packet is sent is when a new Controller becomes available or one or several Controllers advertised in the previous AMP Discover Response or AMP Change Notify packet become unavailable or if the status of a previously advertised Controller changes. The AMP Change Notify carries the new list of the available Controllers. Based on the changes from the previous Change Notify or Discovery Response, the AMP Manager can identify added or deleted controllers or controllers that have changed status.

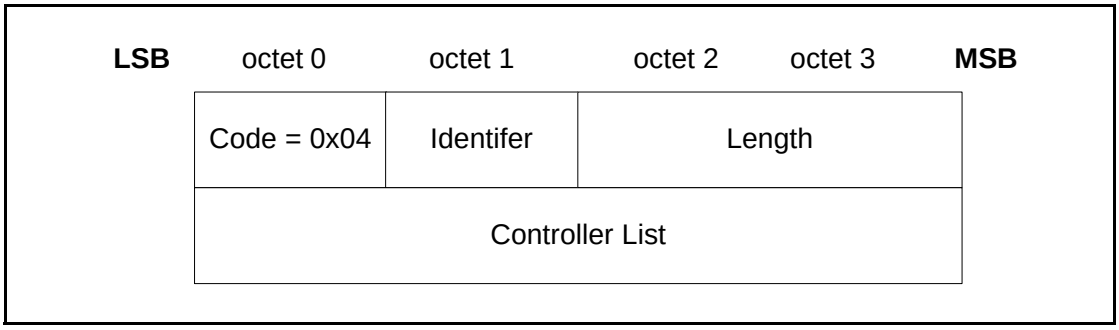



*Figure 3.7: AMP Change Notify Packet*

- *Controller List (3 or more octets)*

  Controller List is variable in length. It consists of 1 or more entries. Each entry is comprised of a Controller ID, a Controller Type and a Controller status (see Section 3.4 for a description of Controller List).

## 3.6 AMP CHANGE RESPONSE (CODE 0X05)

The AMP Change Response packet shall be sent to acknowledge receipt of the AMP Change Notify packet.

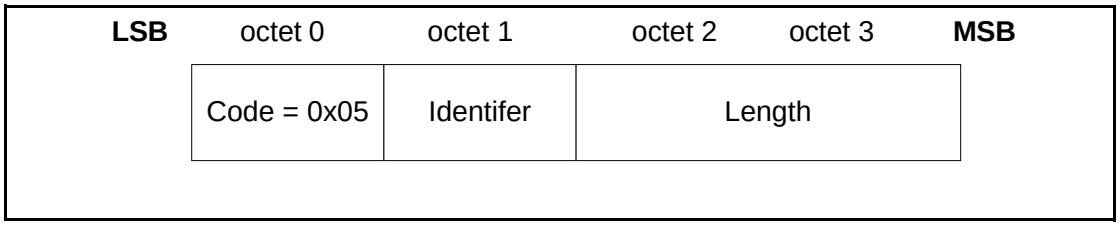



*Figure 3.8: AMP Change Response Packet*

## 3.7 AMP GET INFO REQUEST (CODE 0X06)

To determine if a physical link should be established to a particular AMP Controller, an AMP Manager may need to obtain further information for a remote AMP Controller. For each Controller ID reported in the AMP Discover Response or AMP Change Notify packet the AMP Manager can query this information by issuing an AMP Get Info Request packet. The exception is Controller ID 0. Information for the Primary BR/EDR radio shall not be obtained with an AMP Get Info Request packet.

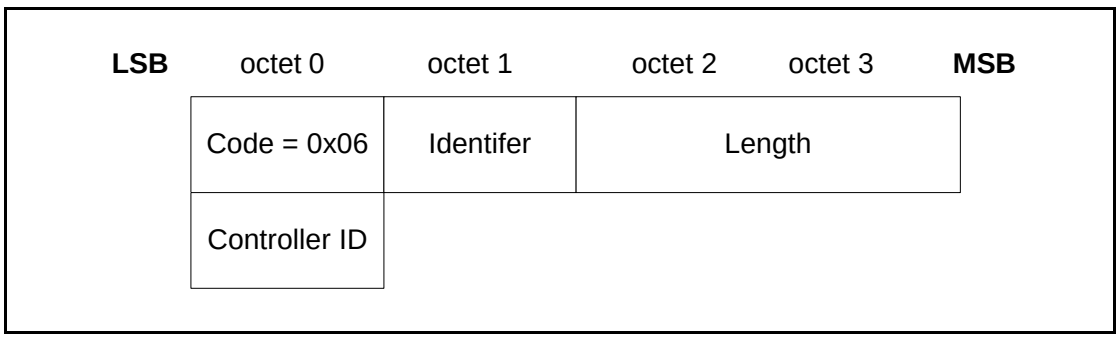



*Figure 3.9: AMP Get Info Request Packet*

The data fields are

- *Controller ID (1 octet)*

  Controller ID is one of the IDs reported in the latest AMP Discover Response, or AMP Change Notify packet received from the peer AMP Manager.

## 3.8 AMP GET INFO RESPONSE (CODE 0X07)

When an AMP Manager receives an AMP Get Info Request packet it shall reply with the AMP Get Info Response packet. This packet carries Controller Information including bandwidth and latency capabilities. The receiver of the

AMP Get Info Response packet can assume that the value of all fields shall remain constant while the remote AMP remains active.

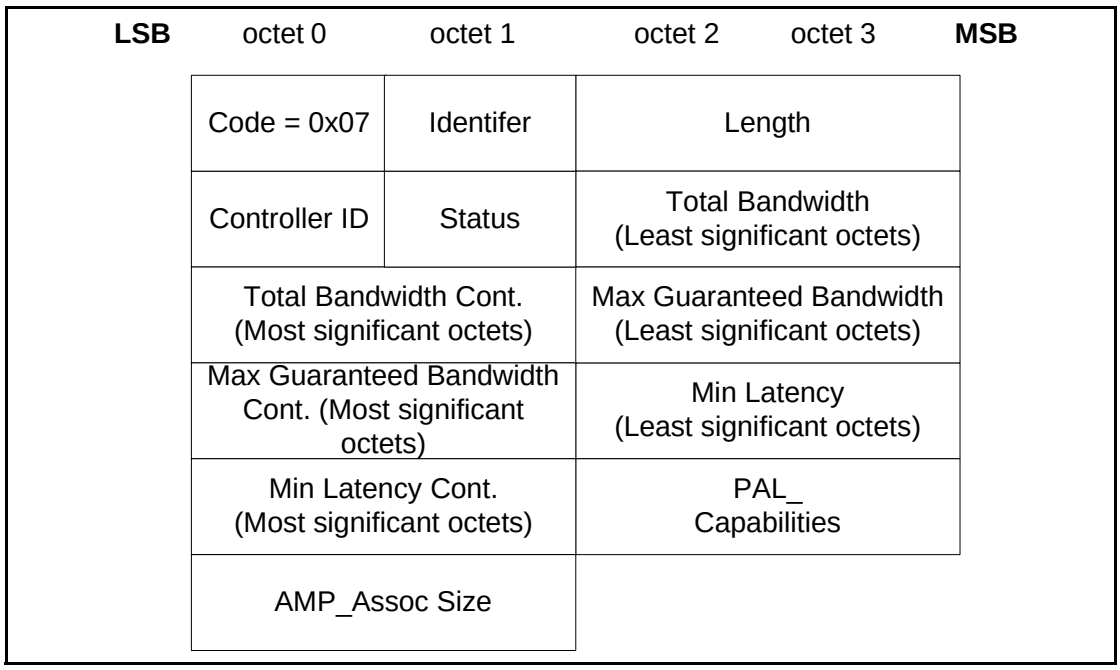



*Figure 3.10: AMP Get Info Response Packet*

- *Controller ID (1 octet)*

  Controller ID is the identifier requested in the AMP Get Info Request.

- *Status (1 octet)*

  The Status field indicates the status of the request

| Status Code | Description |
|---|---|
| 0x00 | Success |
| 0x01 | Invalid Controller ID |
| Other | Reserved |

*Table 3.6: AMP Get Info Response Status values*

An AMP Get Info Response with status of Invalid Controller ID shall be sent by an AMP manager that has received an AMP Get Info Request containing either:

a) A Controller ID that has not been allocated by the AMP manager

b) A Controller ID of 0

If the Status field is set to Invalid Controller ID all subsequent fields in the AMP Get Info Response shall be ignored by the receiver.

- *Total Bandwidth (4 octets)*

The Total Bandwidth field indicates the total data rate in kbits per second (1000 bps) that can be achieved by the Controller for applications. This value shall account for any bandwidth limitations of the Host and the HCI transport if present. The Host is responsible for determining these bandwidth limitations.

- *Max Guaranteed Bandwidth (4 octets)*

  The Maximum Guaranteed Bandwidth field indicates the maximum data rate in kbits per second that the AMP can guarantee for a logical channel. Any request made by an application above this level will be rejected. This value shall account for any bandwidth limitations of the Host and the HCI transport if present. The Host is responsible for determining these bandwidth limitations.

- *Min Latency (4 octets)*

  The Min Latency field indicates the minimum latency in microseconds that the AMP can guarantee for a logical channel. This value shall account for any latency limitations of the Host and the HCI transport if present. The Host is responsible for determining these latency limitations.

- *PAL_Capabilities (2 octets)*

  The PAL_Capabilities field contains the PAL capabilities returned via HCI Read Local AMP Info Command (see [Vol 2 Part E] Section 7.5.8 for more information).

- AMP_Assoc Structure Size (2 octets)

  The maximum size in octets of the requested AMP's AMP_Assoc structure. The value of the AMP_Assoc Structure Size for a given AMP is determined by the controller. If an HCI is being used this value can be retrieved by the Host sending the HCI_Read_Local_AMP_Info command to the Controller.

## 3.9 AMP GET AMP ASSOC REQUEST (CODE 0X08)

To establish a physical link to a particular Controller, an AMP Manager needs to obtain the AMP_Assoc structure for the remote Controller. The remote AMP is identified by its Controller ID as reported in the AMP Discover Response or AMP Change Notify packet. AMP Get AMP Assoc Request packet shall not be used to obtain an AMP_Assoc structure for the Primary BR/EDR Controller.

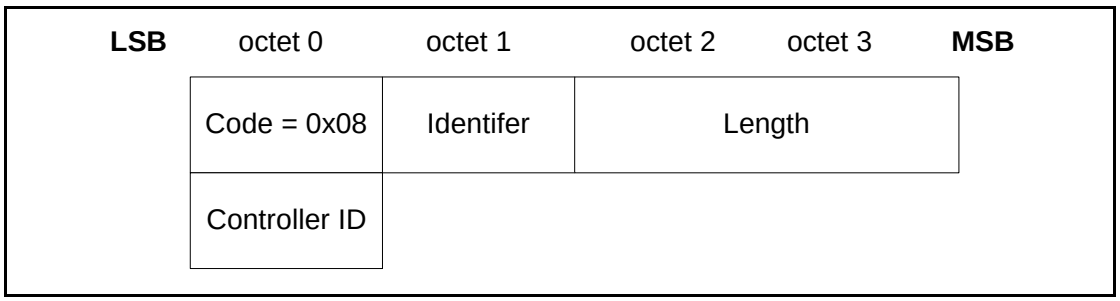



*Figure 3.11: AMP Get AMP Assoc Request Packet*

The data fields are:

- *Controller ID (1 octet)*

  Controller ID is one of the IDs reported in the latest AMP Discover Response or AMP Change Notify packet received from the peer AMP Manager.

## 3.10 AMP GET AMP ASSOC RESPONSE (CODE 0X09)

When an AMP Manager receives an AMP Get AMP Assoc Request packet it shall reply with the AMP Get AMP Assoc Response packet. This packet carries the AMP_Assoc structure. The AMP_Assoc structure is used in creating the physical AMP link. The exact length and format of the AMP_Assoc structure shall be defined in each AMP PAL specification.

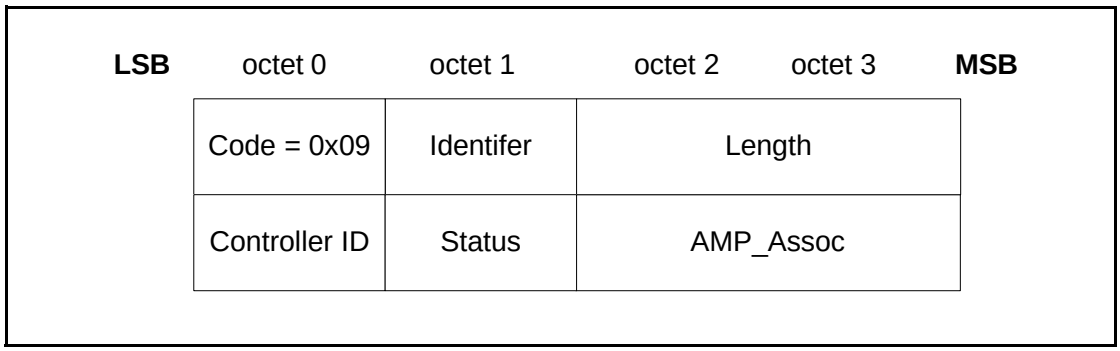



*Figure 3.12: AMP Get AMP Assoc Response Packet*

The data fields are

- *Controller ID (1 octet)*

  Controller ID is the identifier requested in the AMP Get AMP Assoc Request packet.

- *Status (1 octet)*

  The Status field indicates the status of the request.

| Status Code | Description |
|---|---|
| 0x00 | Success |
| 0x01 | Invalid Controller ID |
| Other | Reserved |

*Table 3.7: AMP Get AMP Response Status values*

An AMP Get AMP Assoc Response packet with status of Invalid Controller ID shall be sent by an AMP manager that has received an AMP Get AMP Assoc Request packet containing either

a) A Controller ID that has not been allocated by the AMP manager

b) A Controller ID of 0

If the Status field is set to Invalid Controller ID (0x01) then the AMP_Assoc structure shall not be included in the Get AMPAssoc Response packet.

- *AMP_Assoc structure (from 0 to (A2MP MTU - 6) octets)*

  AMP_Assoc structure is variable in length. The Controller Type of the Controller ID referenced in the AMP Get AMP Assoc Request packet determines the format and the length of the AMP_Assoc structure. The length of the AMP_Assoc structure can be determined by subtracting two from the Length field.

  The length of the AMP_Assoc structure shall not exceed any previously communicated AMP_Assoc length (the AMP_Assoc length is included in the AMP Get Info Response packet. See Section 3.8).

## 3.11 AMP CREATE PHYSICAL LINK REQUEST (CODE 0X0A)

Establishing an AMP physical link requires signaling between the AMP Managers in order to turn the AMP Controller on, if it is not already turned on, and to provide information to set the required state of the AMP Controller, setup addressing, security, etc. When an AMP Manager receives an AMP Create Physical Link request packet it shall initiate its part of the AMP physical link creation and send the AMP Create Physical Link Response packet. AMP Create Physical Link request shall not be used to establish a physical link to the Primary BR/EDR Controller.

For some AMP Controllers it may be required to send additional information to the peer AMP Manager. This information is packaged in a variable size AMP_Assoc structure. The exact length and format of the AMP_Assoc structure shall be defined in each AMP PAL specification.

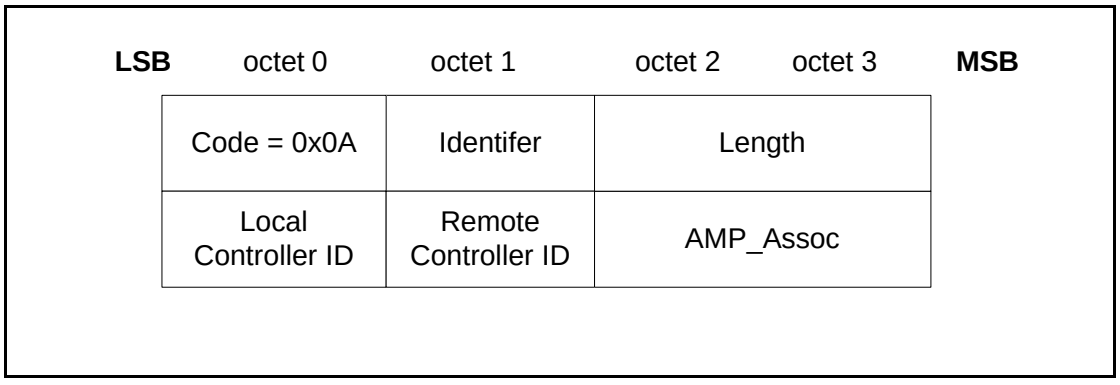



*Figure 3.13: AMP Create Physical Link Request Packet*

- *Local Controller ID (1 octet)*

  Local Controller ID is the ID of the Controller to be used on the device sending the AMP Create Physical Request packet.

- *Remote Controller ID (1 octet)*

  The ID of the AMP Controller to use on the device that receives the AMP Create Physical Link Request packet. The sender of the AMP Create Physical Link Request packet obtains the Remote Controller ID from the latest Discover Response, or the Change Notify packet received from the peer AMP Manager.

- *AMP_Assoc structure (from 0 to (A2MP MTU - 6) octets)*

  AMP_Assoc structure is variable in length. See section 3.10 for a description of AMP_Assoc structure.

## 3.12 AMP CREATE PHYSICAL LINK RESPONSE (CODE 0X0B)

For each AMP Create Physical Link Request the AMP Manager shall reply with an AMP Create Physical Link Response packet. If the device has sent its own AMP Create Physical Link Request to create a physical link with the same AMP Controller a collision has occurred and one of the requests shall be rejected while the other request shall proceed. The algorithm described in Section 3.15 shall be used to determine which request to reject.

When the AMP Manager receives an AMP Create Physical Link Request packet it shall try to perform the requested actions and send a response identifying the status of the operation.

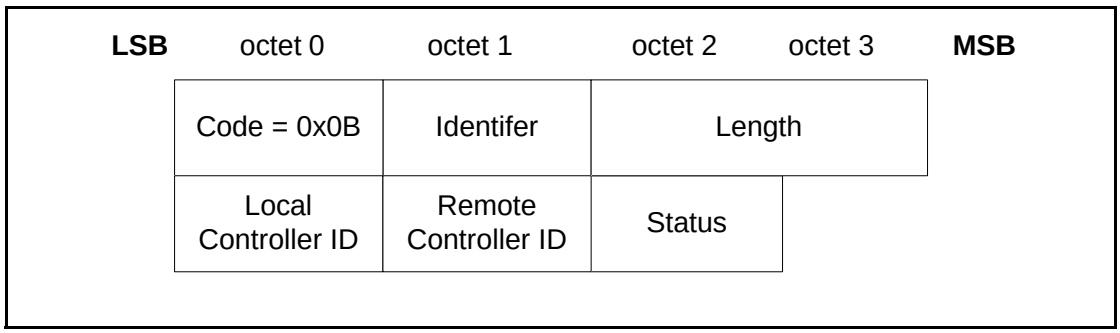



*Figure 3.14: AMP Create Physical Link Response Packet*

The data fields are

- *Local Controller ID (1 octet)*

  This field contains the Controller ID local to the sender of the AMP Create Physical Link Response packet. The ID value is the same as the Remote Controller ID that was in the associated AMP Create Physical Link Request packet.

- *Responder Controller ID (1 octet)*

This field contains the Controller ID used by the receiver of the AMP Create Physical Link Response packet. The ID value is the same as the Local Controller ID that was in the associated AMP Create Physical Link Request packet.

- *Status (1 octet)*

  The Status field indicates the result of the request.

| Status Code | Description |
|---|---|
| 0x00 | Success - Link creation is started |
| 0x01 | Invalid Controller ID. |
| 0x02 | Failed - Unable to start link creation |
| 0x03 | Failed - Collision Occurred |
| 0x04 | Failed - AMP Disconnected Physical Link Request packet received |
| 0x05 | Failed - Physical Link Already Exists |
| 0x06 | Failed - Security violation |
| Other | Reserved |

*Table 3.8: AMP Create Physical Link Response Status values*

An AMP Create Physical Link Response with status of Invalid Controller ID shall be sent by an AMP manager that has received an AMP Create Physical Link Request containing either

a) A Controller ID that has not been allocated by the AMP manager

b) A Controller ID of 0

An AMP Create Physical Link Response with status of Failed - Security Violation shall be sent by an AMP manager that has received an AMP Create Physical Link Request without mutual authentication being performed over the BR/EDR Controller and without encryption being enabled on the BR/EDR Controller.

## 3.13 AMP DISCONNECT PHYSICAL LINK REQUEST (CODE 0X0C)

AMP Disconnect Physical Link request is an optional request that can be used by either initiator or responder to either abort the creation of an AMP physical link or disconnect an existing Physical Link.

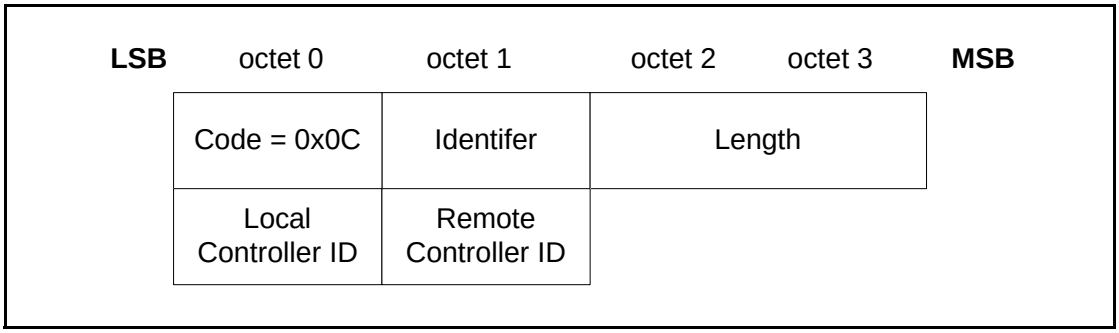



*Figure 3.15: AMP Disconnect Physical Link Request Packet*

The data fields are:

- *Local Controller ID (1 octet)*

  Local Controller ID is the ID of the Controller on the device sending the AMP Disconnect Physical Link Request packet.

- *Remote Controller ID (1 octet)*

  The ID of the AMP Controller on the device that receives the AMP Disconnect Physical Link Request packet.

## 3.14 AMP DISCONNECT PHYSICAL LINK RESPONSE (CODE 0X0D)

If an AMP Manager receives an AMP Disconnect Physical Link request packet while creation of a physical link is outstanding, it shall abort the operation. In devices that support HCI the Host may send an HCI_Disconnect_Physical_Link command to cause the controller to abort the creation of a Physical Link.

If an AMP Disconnect Physical Link request packet is received from a device that has an AMP Create Physical Link request packet outstanding, the receiver shall send both an AMP Create Physical Link Response packet with a status of 'Failed - AMP Disconnected Physical Link Request packet received' and an AMP Disconnect Physical Link response packet with a status of 'Success'.

If an AMP manager receives an AMP Disconnect Physical Link request packet when creation of a physical link is not outstanding and a Physical Link does not exist for the given local and remote AMP IDs it shall send an AMP Disconnect Physical Link Response with a status of 'Failed - No Physical Link exists and no Physical Link creation is in progress'.

The status of 'Invalid Controller ID' shall only be used if the receiver of the AMP Disconnect Request packet does not have an AMP with an ID that matches the Remote Controller ID in the received AMP Disconnect Request packet.

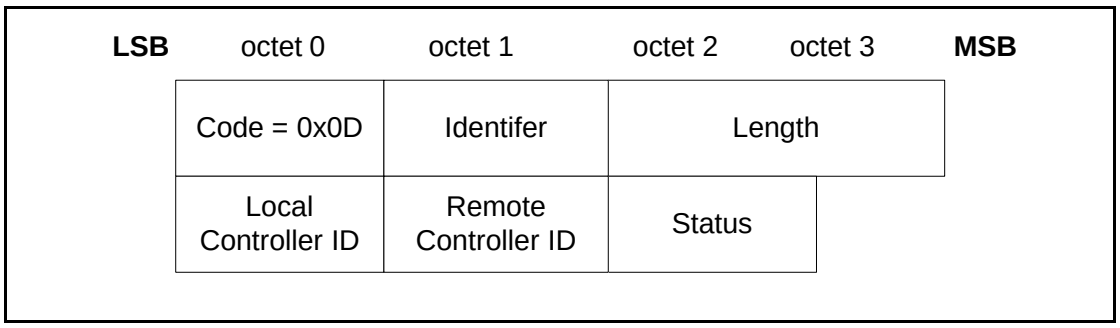



*Figure 3.16: AMP Disconnect Physical Link Response Packet*

The data fields are:

- *Local Controller ID (1 octet)*

  This field contains the Controller ID local to the sender of the AMP Disconnect Physical Link Response packet. The ID value is the same as the Remote Controller ID that was in the associated AMP Disconnect Physical Link Request packet.

- *Remote Controller ID (1 octet)*

  This field contains the Controller ID used by the receiver of the AMP Disconnect Physical Link Response packet. The ID value is the same as the Local Controller ID that was in the associated AMP Disconnect Physical Link Request packet.

- *Status (1 octet)*

  The Status field indicates the result of the request.

| Status Code | Description |
|---|---|
| 0x00 | Success |
| 0x01 | Invalid Controller ID. |
| 0x02 | Failed - No Physical Link exists and no Physical Link creation is in progress. |
| Other | Reserved |

*Table 3.9: AMP Disconnect Physical Link Response Status values*

## 3.15 CREATE PHYSICAL LINK COLLISION RESOLUTION

If two devices simultaneously attempt to create a physical link a collision will occur. In this situation one of the requests shall be canceled while the other request shall proceed. Both devices must know which request will be rejected. The following algorithm shall be used by both devices to determine which request to reject.

1. Set i=0 (representing the least significant octet of the BD_ADDR)

2. Compare the least significant octet, octet[i] of the BD_ADDR, of both devices. If the octets are not equal go to step 4.
3. Increment i by 1. Go to step 2.
4. The device with the larger BD_ADDR octet shall send an AMP Create Physical Link Response with the Status field set to "Failed - Collision Occurred."

## 3.16 RESPONSE TIMEOUT

The Response Timeout is used to monitor the response to request packets. The exact timeout value is implementation dependent and may be chosen within the range of 15 - 60 seconds. The recommended value is 20 seconds. If a response to a request packet is not received within the Response Timeout then the ACL link shall be disconnected.

## 3.17 UNEXPECTED BR/EDR PHYSICAL LINK DISCONNECT

It is possible that the BR/EDR link between two devices might unexpectedly disconnect (Link Supervision Timeout) while AMP links between the same two devices remain active. This situation is problematic since the BR/EDR link is used for L2CAP and AMP Manager signaling between the two devices.

When the BR/EDR link with a remote device unexpectedly disconnects due to Link Supervision Timeout the AMP manager shall disconnect all AMP physical links to the same remote device.

# FIGURES

Figure 2.1: Overview of the Lower Software Layers ..... 443
Figure 2.2: AMP Manager Peers ..... 444
Figure 3.1: AMP Manager Protocol Packet Format ..... 448
Figure 3.2: AMP Command Reject Packet ..... 450
Figure 3.3: AMP Discover Request Packet ..... 450
Figure 3.4: Example A2MP Extended Feature Mask ..... 451
Figure 3.5: AMP Discover Response Packet ..... 452
Figure 3.6: Controller list format ..... 453
Figure 3.7: AMP Change Notify Packet ..... 455
Figure 3.8: AMP Change Response Packet ..... 456
Figure 3.9: AMP Get Info Request Packet ..... 456
Figure 3.10: AMP Get Info Response Packet ..... 457
Figure 3.11: AMP Get AMP Assoc Request Packet ..... 458
Figure 3.12: AMP Get AMP Assoc Response Packet ..... 459
Figure 3.13: AMP Create Physical Link Request Packet ..... 460
Figure 3.14: AMP Create Physical Link Response Packet ..... 461
Figure 3.15: AMP Disconnect Physical Link Request Packet ..... 463
Figure 3.16: AMP Disconnect Physical Link Response Packet ..... 464

# TABLES

Table 2.1: AMP Manager Channel configuration parameters ..... 444
Table 2.2: AMP Manager Channel Enhanced Retransmission Mode parameters ..... 445
Table 3.1: Codes ..... 448
Table 3.2: Identifier ..... 449
Table 3.3: AMP Command Reject reasons ..... 450
Table 3.4: A2MP Extended Feature Mask ..... 451
Table 3.5: Controller Status ..... 455
Table 3.6: AMP Get Info Response Status values ..... 457
Table 3.7: AMP Get AMP Response Status values ..... 459
Table 3.8: AMP Create Physical Link Response Status values ..... 462
Table 3.9: AMP Disconnect Physical Link Response Status values ..... 464

# ATTRIBUTE PROTOCOL (ATT)

***This specification defines the Attribute Protocol; a protocol for discovering, reading, and writing attributes on a peer device***


Ford Ex. 1052

# CONTENTS

**1 Introduction ....473**
1.1 Scope ....473
1.2 Conformance ....473
**2 Protocol Overview ....474**
**3 Protocol Requirements ....475**
3.1 Introduction ....475
3.2 Basic Concepts ....475
3.2.1 Attribute Type ....475
3.2.2 Attribute Handle ....476
3.2.3 Attribute Handle Grouping ....476
3.2.4 Attribute Value ....476
3.2.5 Attribute Permissions ....477
3.2.6 Control-Point Attributes ....478
3.2.7 Protocol Methods ....478
3.2.8 Exchanging MTU Size ....479
3.2.9 Long Attribute Values ....479
3.2.10 Atomic Operations ....479
3.3 Attribute PDU ....480
3.3.1 Attribute PDU Format ....481
3.3.2 Sequential Protocol ....482
3.3.3 Transaction ....482
3.4 Attribute Protocol PDUs ....483
3.4.1 Error Handling ....483
3.4.1.1 Error Response ....483
3.4.2 MTU Exchange ....485
3.4.2.1 Exchange MTU Request ....485
3.4.2.2 Exchange MTU Response ....486
3.4.3 Find Information ....487
3.4.3.1 Find Information Request ....487
3.4.3.2 Find Information Response ....488
3.4.3.3 Find By Type Value Request ....489
3.4.3.4 Find By Type Value Response ....490
3.4.4 Reading Attributes ....491
3.4.4.1 Read By Type Request ....491
3.4.4.2 Read By Type Response ....493
3.4.4.3 Read Request ....494
3.4.4.4 Read Response ....494
3.4.4.5 Read Blob Request ....495

3.4.4.6 Read Blob Response ..... 496
3.4.4.7 Read Multiple Request ..... 497
3.4.4.8 Read Multiple Response ..... 498
3.4.4.9 Read by Group Type Request ..... 498
3.4.4.10 Read by Group Type Response ..... 501
3.4.5 Writing Attributes ..... 501
3.4.5.1 Write Request ..... 501
3.4.5.2 Write Response ..... 503
3.4.5.3 Write Command ..... 503
3.4.5.4 Signed Write Command ..... 504
3.4.6 Queued Writes ..... 505
3.4.6.1 Prepare Write Request ..... 505
3.4.6.2 Prepare Write Response ..... 507
3.4.6.3 Execute Write Request ..... 508
3.4.6.4 Execute Write Response ..... 509
3.4.7 Server Initiated ..... 509
3.4.7.1 Handle Value Notification ..... 509
3.4.7.2 Handle Value Indication ..... 509
3.4.7.3 Handle Value Confirmation ..... 510
3.4.8 Attribute Opcode Summary ..... 510
3.4.9 Attribute PDU Response Summary ..... 513
**4 Security Considerations ..... 516**
**5 Acronyms and Abbreviations ..... 517**

# 1 INTRODUCTION

## 1.1 SCOPE

The attribute protocol allows a device referred to as the server to expose a set of attributes and their associated values to a peer device referred to as the client. These attributes exposed by the server can be discovered, read, and written by a client, and can be indicated and notified by the server.

## 1.2 CONFORMANCE

If conformance to this protocol is claimed, all capabilities indicated as mandatory for this protocol shall be supported in the specified manner (process-mandatory). This also applies to all optional and conditional capabilities for which support is indicated. All mandatory capabilities, and optional and conditional capabilities for which support is indicated, are subject to verification as part of the Bluetooth qualification program.

# 2 PROTOCOL OVERVIEW

The attribute protocol defines two roles; a server role and a client role. It allows a server to expose a set of attributes to a client that are accessible using the attribute protocol.

An attribute is a discrete value that has the following three properties associated with it:

a) attribute type, defined by a UUID

b) attribute handle

c) a set of permissions that are defined by each higher layer specification that utilizes the attribute; these permissions cannot be accessed using the Attribute Protocol.

The attribute type specifies what the attribute represents. Bluetooth SIG defined attribute types are defined in the Bluetooth SIG assigned numbers page, and used by an associated higher layer specification. Non-Bluetooth SIG attribute types may also be defined.

The attribute handle uniquely identifies an attribute on a server, allowing a client to reference the attribute in read or write requests; see Section 3.4.4, Section 3.4.5, and Section 3.4.6. It allows a client to uniquely identify the attribute being notified or indicated, see Section 3.4.7. Clients are able to discover the handles of the server's attributes; see Section 3.4.3. Permissions may be applied to an attribute to prevent applications from obtaining or altering an attribute's value. An attribute may be defined by a higher layer specification to be readable or writable or both, and may have additional security requirements. For more information, see Section 3.2.5.

A client may send attribute protocol requests to a server, and the server shall respond to all requests that it receives. A device can implement both client and server roles, and both roles can function concurrently in the same device and between the same devices. There shall be only one instance of a server on each Bluetooth device; this implies that the attribute handles shall be identical for all supported bearers. For a given client, the server shall have one set of attributes. The server can support multiple clients. Note: multiple services may be exposed on a single server by allocating separate ranges of handles for each service. The discovery of these handle ranges is defined by a higher layer specification.

The attribute protocol has notification and indication capabilities that provide an efficient way of sending attribute values to a client without the need for them to be read; see Section 3.3.

# 3 PROTOCOL REQUIREMENTS

## 3.1 INTRODUCTION

Each attribute has an attribute type that identifies, by means of a UUID (Universally Unique IDentifier), what the attribute represents so that a client can understand the attributes exposed by a server. Each attribute has an attribute handle that is used for accessing the attribute on a server, as well as an attribute value.

An attribute value is accessed using its attribute handle. The attribute handles are discovered by a client using attribute protocol PDUs (Protocol Data Unit). Attributes that have the same attribute type may exist more than once in a server. Attributes also have a set of permissions that controls whether they can be read or written, or whether the attribute value shall be sent over an encrypted link. Security aspects of the attribute protocol are defined in Section 4.

## 3.2 BASIC CONCEPTS

### 3.2.1 Attribute Type

A universally unique identifier (UUID) is used to identify every attribute type. A UUID is considered unique over all space and time. A UUID can be independently created by anybody and distributed or published as required. There is no central registry for UUIDs, as they are based off a unique identifier that is not duplicated. The attribute protocol allows devices to identify attribute types using UUIDs regardless of the local handle used to identify them in a read or write request.

A UUID is a 128-bit value[1]. To improve efficiency, common or frequently used UUIDs may be shortened to 16 bits. The Bluetooth Base UUID is defined that all 16-bit Attribute UUIDs use as a base. The Bluetooth_Base_UUID is:

00000000-0000-1000-8000-00805F9B34FB

To convert a 16-bit Attribute UUID into a full 128-bit UUID the following arithmetic operation can be performed

128-bit UUID = 16-bit Attribute UUID * $2^{96}$ + Bluetooth_Base_UUID

Or, to put it more simply, the 16-bit Attribute UUID replaces the x's in the following:

0000xxxx-0000-1000-8000-00805F9B34FB

1. The format of a UUID is specified in ITU-T Rec. X.667, alternatively known as ISO/IEC 9834-8:2005.

For example, the 16-bit Attribute UUID of 0x1234 is equivalent to the 128-bit UUID of

00001234-0000-1000-8000-00805F9B34FB

A 16-bit Attribute UUID can be directly compared with another 16-bit Attribute UUID without conversion to a 128-bit UUID first. However, whenever comparing a 16-bit Attribute UUID with a 128-bit UUID, the 16-bit Attribute UUID shall be converted to a 128-bit UUID first.

Applications shall not convert a 128-bit UUID to a 16-bit UUID for comparison as there is no guarantee that the 128-bit UUID is based on the Bluetooth Base UUID; it could be a vendor defined 128-bit UUID.

16-bit UUIDs are assigned by the Bluetooth SIG and published in the Bluetooth Assigned Numbers page.

Note: The 16-bit Attribute UUIDs use the same namespace as SDP 16-bit UUIDs.

### 3.2.2 Attribute Handle

An attribute handle is a 16-bit value that is assigned by each server to its own attributes to allow a client to reference those attributes. An attribute handle shall not be reused while an ATT Bearer exists between a client and its server.

Attribute handles on any given server shall have unique, non-zero values. Attributes are ordered by attribute handle.

An attribute handle of value 0x0000 is reserved, and shall not be used. An attribute handle of value 0xFFFF is known as the maximum attribute handle.

Note: Attributes can be added or removed while an ATT Bearer is active, however, an attribute that has been removed cannot be replaced by another attribute with the same handle while an ATT Bearer is active.

### 3.2.3 Attribute Handle Grouping

Grouping is defined by a specific attribute placed at the beginning of a range of other attributes that are grouped with that attribute, as defined by a higher layer specification. Clients can request the first and last handles associated with a group of attributes.

### 3.2.4 Attribute Value

An attribute value is an octet array that may be either fixed or variable length. For example, it can be a one octet value, or a four octet integer, or a variable length string. An attribute may contain a value that is too large to transmit in a single PDU and can be sent using multiple PDUs. The values that are transmit-

ted are opaque to the attribute protocol. The encoding of these octet arrays is defined by the attribute type.

When transmitting attribute values in a request, a response, a notification or an indication, the attribute value length is not sent in any field of the PDU. The length of a variable length field in the PDU is implicitly given by the length of the packet that carries this PDU. This implies that:

a) only one attribute value can be placed in a single request, response, notification or indication unless the attribute values have lengths known by both the server and client, as defined by the attribute type.

b) This attribute value will always be the only variable length field of a request, response, notification or indication.

c) The bearer protocol (e.g. L2CAP) preserves datagram boundaries.

Note: Some responses include multiple attribute values, for example when client requests multiple attribute reads. For the client to determine the attribute value boundaries, the attribute values must have a fixed size defined by the attribute type.

### 3.2.5 Attribute Permissions

An attribute has a set of permission values associated with it. The permissions associated with an attribute specifies that it may be read and/or written. The permissions associated with the attribute specifies the security level required for read and/or write access, as well as notification and/or indication. The permissions of a given attribute are defined by a higher layer specification, and are not discoverable using the attribute protocol.

If access to a secure attribute requires an authenticated link, and the client is not already authenticated with the server with sufficient security, then an error response shall be sent with the error code «Insufficient Authentication». When a client receives this error code it may try to authenticate the link, and if the authentication is successful, it can then access the secure attribute.

If access to a secure attribute requires an encrypted link, and the link is not encrypted, then an error response shall be sent with the error code «Insufficient Encryption». When a client receives this error code it may try to encrypt the link and if the encryption is successful, it can then access the secure attribute.

If access to a secure attribute requires an encrypted link, and the link is encrypted but with an encryption key size that is too short for the level of security required, then an error response shall be sent with the error code «Insufficient Encryption Key Size». When a client receives this error code it may try to encrypt the link with a larger key size, and if the encryption is successful, it can then access the secure attribute.

Attribute permissions are a combination of access permissions, authentication permissions and authorization permissions.

The following access permissions are possible:

- Readable
- Writeable
- Readable and writable

The following authentication permissions are possible:

- Authentication Required
- No Authentication Required

The following authorization permissions are possible:

- Authorization Required
- No Authorization Required

Access permissions are used by a server to determine if a client can read and/or write an attribute value.

Authentication permissions are used by a server to determine if an authenticated physical link is required when a client attempts to access an attribute. Authentication permissions are also used by a server to determine if an authenticated physical link is required before sending a notification or indication to a client.

Authorization permissions determine if a client needs to be authorized before accessing an attribute value.

### 3.2.6 Control-Point Attributes

Attributes that cannot be read, but can only be written, notified or indicated are called control-point attributes. These control-point attributes can be used by higher layers to enable device specific procedures, for example the writing of a command or the indication when a given procedure on a device has completed.

### 3.2.7 Protocol Methods

The attribute protocol uses methods defined in Section 3.4 to find, read, write, notify, and indicate attributes. A method is categorized as either a request, a response, a command, a notification, an indication or a confirmation method; see Section 3.3. Some attribute protocol PDUs can also include an Authentication Signature, to allow authentication of the originator of this PDU without requiring encryption. The method and signed bit are known as the opcode.

### 3.2.8 Exchanging MTU Size

ATT_MTU is defined as the maximum size of any packet sent between a client and a server. A higher layer specification defines the default ATT_MTU value.

The client and server may optionally exchange the maximum size of packet that can be received using the Exchange MTU Request and Response PDUs. Both devices then use the minimum of these exchanged values for all further communication (see Section 3.4.2).

A device that is acting as a server and client at the same time shall use the same value for Client Rx MTU and Server Rx MTU.

The ATT_MTU value is a per ATT Bearer value. Note: A device with multiple ATT Bearers may have a different ATT_MTU value for each ATT Bearer.

### 3.2.9 Long Attribute Values

The longest attribute that can be sent in a single packet is (ATT_MTU−1) octets in size. At a minimum, the Attribute Opcode is included in an Attribute PDU.

An attribute value may be defined to be larger than (ATT_MTU−1) octets in size. These attributes are called long attributes.

To read the entire value of an attributes larger than (ATT_MTU−1) octets, the read blob request is used. It is possible to read the first (ATT_MTU−1) octets of a long attribute value using the read request.

To write the entire value of an attribute larger than (ATT_MTU−3) octets, the prepare write request and execute write request is used. It is possible to write the first (ATT_MTU−3) octets of a long attribute value using the write request.

It is not possible to determine if an attribute value is longer than (ATT_MTU−3) octets using this protocol. A higher layer specification will state that a given attribute can have a maximum length larger than (ATT_MTU−3) octets.

The maximum length of an attribute value shall be 512 octets.

Note: The protection of an attribute value changing when reading the value using multiple attribute protocol PDUs is the responsibility of the higher layer.

### 3.2.10 Atomic Operations

The server shall treat each request or command as an atomic operation that cannot be affected by another client sending a request or command at the same time. If a link is disconnected for any reason (user action or loss of the radio link), the value of any modified attribute is the responsibility of the higher specification.

Long attributes cannot be read or written in a single atomic operation.

## 3.3 ATTRIBUTE PDU

Attribute PDUs are one of six method types:

- Requests—PDUs sent to a server by a client, and invoke responses.
- Responses—PDUs sent to a client in response to a request to a server.
- Commands—PDUssent to a server by a client.
- Notifications—Unsolicited PDUs sent to a client by a server.
- Indications—Unsolicited PDUs sent to a client by a server, and invoke confirmations.
- Confirmations—PDUs sent to an attribute server to confirm receipt of an indication by a client.

A server shall be able to receive and properly respond to the following requests:

- *Find Information Request*
- Read Request

Support for all other PDU types in a server can be specified in a higher layer specification, see Section 3.4.8.

If a client sends a request, then the client shall support all possible responses PDUs for that request.

If a server receives a request that it does not support, then the server shall respond with the *Error Response* with the Error Code «Request Not Supported», with the Attribute Handle In Error set to 0x0000.

If a server receives a command that it does not support, indicated by the Command Flag of the PDU set to one, then the server shall ignore the Command.

If the server receives an invalid request–for example, the PDU is the wrong length–then the server shall respond with the *Error Response* with the Error Code «Invalid PDU», with the Attribute Handle In Error set to 0x0000.

If a server does not have sufficient resources to process a request, then the server shall respond with the *Error Response* with the Error Code «Insufficient Resources», with the Attribute Handle In Error set to 0x0000.

If a server cannot process a request because an error was encountered during the processing of this request, then the server shall respond with the *Error Response* with the Error Code «Unlikely Error», with the Attribute Handle In Error set to 0x0000.

### 3.3.1 Attribute PDU Format

Attribute PDUs has the following format:

| Name | Size (octets) | Description |
|---|---|---|
| Attribute Opcode | 1 | The attribute PDU operation code<br>bit7: Authentication Signature Flag<br>bit6: Command Flag<br>bit5-0: Method |
| Attribute Parameters | 0 to (ATT_MTU - X) | The attribute PDU parameters<br>X = 1 if Authentication Signature Flag of the Attribute Opcode is 0<br>X = 13 if Authentication Signature Flag of the Attribute Opcode is 1 |
| Authentication Signature | 0 or 12 | Optional authentication signature for the Attribute Opcode and Attribute Parameters |

*Table 3.1: Format of Attribute PDU*

Multi-octet fields within the attribute protocol shall be sent least significant octet first (little endian) with the exception of the Attribute Value field. The endianness of the Attribute Value field is defined by a higher layer specification.

The Attribute Opcode is composed of three fields, the Authentication Signature Flag, the Command Flag, and the Method. The Method is a 6-bit value that determines the format and meaning of the Attribute Parameters.

If the Authentication Signature Flag of the Attribute Opcode is set to one, the Authentication Signature value shall be appended to the end of the attribute PDU, and X is 13. If the Authentication Signature Flag of the Attribute Opcode is set to zero, the Authentication Signature value shall not be appended, and X is 1.

The Authentication Signature field is calculated as defined in Security Manager (see [Vol. 3] Part H, Section 2.4.5). This value provides an Authentication Signature for the following values in this order: Attribute Opcode, Attribute Parameters.

An Attribute PDU that includes an Authentication Signature should not be sent on an encrypted link. Note: an encrypted link already includes authentication data on every packet and therefore adding more authentication data is not required.

If the Command Flag of the Attribute Opcode is set to one, the PDU shall be considered to be a Command.

Only the Write Command may include an Authentication Signature:.

### 3.3.2 Sequential Protocol

Many attribute protocol PDUs use a sequential request-response protocol.

Once a client sends a request to a server, that client shall send no other request to the same server until a response PDU has been received.

Indications sent from a server also use a sequential indication-confirmation protocol. No other indications shall be sent to the same client from this server until a confirmation PDU has been received. The client, however, is free to send commands and requests prior to sending a confirmation.

For notifications, which do not have a response PDU, there is no flow control and a notification can be sent at any time.

Commands that do not require a response do not have any flow control. Note: a server can be flooded with commands, and a higher layer specification can define how to prevent this from occurring.

Commands and notifications that are received but cannot be processed, due to buffer overflows or other reasons, shall be discarded. Therefore, those PDUs must be considered to be unreliable.

Note: Flow control for each client and a server is independent.

Note: It is possible for a server to receive a request, send one or more notifications, and then the response to the original request. The flow control of requests is not affected by the transmission of the notifications.

Note: It is possible for a server to receive a request and then a command before responding to the original request. The flow control of requests is not affected by the transmission of commands.

Note: It is possible for a notification from a server to be sent after an indication has been sent but the confirmation has not been received. The flow control of indications is not affected by the transmission of notifications.

Note: It is possible for a client to receive an indication from a server and then send a request or command to that server before sending the confirmation of the original indication.

### 3.3.3 Transaction

An attribute protocol request and response or indication-confirmation pair is considered a single transaction. A transaction shall always be performed on one ATT Bearer, and shall not be split over multiple ATT Bearer.

On the client, a transaction shall start when the request is sent by the client. A transaction shall complete when the response is received by the client.

On a server, a transaction shall start when a request is received by the server. A transaction shall complete when the response is sent by the server.

On a server, a transaction shall start when an indication is sent by the server. A transaction shall complete when the confirmation is received by the server.

On a client, a transaction shall start when an indication is received by the client. A transaction shall complete when the confirmation is sent by the client.

A transaction not completed within 30 seconds shall time out. Such a transaction shall be considered to have failed and the local higher layers shall be informed of this failure. No more attribute protocol requests, commands, indications or notifications shall be sent to the target device on this ATT Bearer.

Note: To send another attribute protocol PDU, a new ATT Bearer must be established between these devices. The existing ATT Bearer may need to be disconnected or the bearer terminated before the new ATT Bearer is established.

If the ATT Bearer is disconnected during a transaction, then the transaction shall be considered to be closed, and any values that were being modified on the server will be in an undetermined state, and any queue that was prepared by the client using this ATT Bearer shall be cleared.

Note: Each Prepare Write Request is a separate request and is therefore a separate transaction.

Note: Each Read Blob Request is a separate request and is therefore a separate transaction.

## 3.4 ATTRIBUTE PROTOCOL PDUS

### 3.4.1 Error Handling

#### *3.4.1.1 Error Response*

The *Error Response* is used to state that a given request cannot be performed, and to provide the reason.

Note: The Write Command does not generate an Error Response.

| Parameter | Size (octets) | Description |
|---|---|---|
| Attribute Opcode | 1 | 0x01 = Error Response |
| Request Opcode In Error | 1 | The request that generated this error response |

*Table 3.2: Format of Error Response*

| Parameter | Size (octets) | Description |
|---|---|---|
| Attribute Handle In Error | 2 | The attribute handle that generated this error response |
| Error Code | 1 | The reason why the request has generated an error response |

*Table 3.2: Format of Error Response*

The Request Opcode In Error parameter shall be set to the Attribute Opcode of the request that generated this error.

The Attribute Handle In Error parameter shall be set to the attribute handle in the original request that generated this error. If there was no attribute handle in the original request or if the request is not supported, then the value 0x0000 shall be used for this field.

The Error Code parameter shall be set to one of the following values:

| Name | Error Code | Description |
|---|---|---|
| Invalid Handle | 0x01 | The attribute handle given was not valid on this server. |
| Read Not Permitted | 0x02 | The attribute cannot be read. |
| Write Not Permitted | 0x03 | The attribute cannot be written. |
| Invalid PDU | 0x04 | The attribute PDU was invalid. |
| Insufficient Authentication | 0x05 | The attribute requires authentication before it can be read or written. |
| Request Not Supported | 0x06 | Attribute server does not support the request received from the client. |
| Invalid Offset | 0x07 | Offset specified was past the end of the attribute. |
| Insufficient Authorization | 0x08 | The attribute requires authorization before it can be read or written. |
| Prepare Queue Full | 0x09 | Too many prepare writes have been queued. |
| Attribute Not Found | 0x0A | No attribute found within the given attribute handle range. |
| Attribute Not Long | 0x0B | The attribute cannot be read or written using the Read Blob Request |
| Insufficient Encryption Key Size | 0x0C | The Encryption Key Size used for encrypting this link is insufficient. |
| Invalid Attribute Value Length | 0x0D | The attribute value length is invalid for the operation. |

*Table 3.3: Error Codes*

| Name | Error Code | Description |
|---|---|---|
| Unlikely Error | 0x0E | The attribute request that was requested has encountered an error that was unlikely, and therefore could not be completed as requested. |
| Insufficient Encryption | 0x0F | The attribute requires encryption before it can be read or written. |
| Unsupported Group Type | 0x10 | The attribute type is not a supported grouping attribute as defined by a higher layer specification. |
| Insufficient Resources | 0x11 | Insufficient Resources to complete the request |
| Reserved | 0x012 – 0x7F | Reserved for future use. |
| Application Error | 0x80 – 0xFF | Application error code defined by a higher layer specification. |

*Table 3.3: Error Codes*

If an error code is received in the *Error Response* that is not understood by the client, for example an error code that was reserved for future use that is now being used in a future version of this specification, then the *Error Response* shall still be considered to state that the given request cannot be performed for an unknown reason.

### 3.4.2 MTU Exchange

#### *3.4.2.1 Exchange MTU Request*

The *Exchange MTU Request* is used by the client to inform the server of the client's maximum receive MTU size and request the server to respond with its maximum receive MTU size.

| Parameter | Size (octets) | Description |
|---|---|---|
| Attribute Opcode | 1 | 0x02 = Exchange MTU Request |
| Client Rx MTU | 2 | Client receive MTU size |

*Table 3.4: Format of Exchange MTU Request*

The Client Rx MTU shall be greater than or equal to the default ATT_MTU.

This request shall only be sent once during a connection by the client. The Client Rx MTU parameter shall be set to the maximum size of the attribute protocol PDU that the client can receive.

### 3.4.2.2 Exchange MTU Response

The *Exchange MTU Response* is sent in reply to a received *Exchange MTU Request*.

| Parameter | Size (octets) | Description |
|---|---|---|
| Attribute Opcode | 1 | 0x03 = Exchange MTU Response |
| Server Rx MTU | 2 | Attribute server receive MTU size |

*Table 3.5: Format of Exchange MTU Response*

The Server Rx MTU shall be greater than or equal to the default ATT_MTU.

The Server Rx MTU parameter shall be set to the maximum size of the attribute protocol PDU that the server can receive.

The server and client shall set ATT_MTU to the minimum of the Client Rx MTU and the Server Rx MTU. The size is the same to ensure that a client can correctly detect the final packet of a long attribute read.

This ATT_MTU value shall be applied in the server after this response has been sent and before any other attribute protocol PDU is sent.

This ATT_MTU value shall be applied in the client after this response has been received and before any other attribute protocol PDU is sent.

If either Client Rx MTU or Service Rx MTU are incorrectly less than the default ATT_MTU, then the ATT_MTU shall not be changed and the ATT_MTU shall be the default ATT_MTU.

If a device is both a client and a server, the following rules shall apply:

1. A device's *Exchange MTU Request* shall contain the same MTU as the device's *Exchange MTU Response* (i.e. the MTU shall be symmetric).
2. If MTU is exchanged in one direction, that is sufficient for both directions.
3. It is permitted, (but not necessary - see 2.) to exchange MTU in both directions, but the MTUs shall be the same in each direction (see 1.)
4. If an Attribute Protocol Request is received after the MTU Exchange Request is sent and before the MTU Exchange Response is received, the associated Attribute Protocol Response shall use the default MTU. Figure 3.1 shows an example that is covered by this rule. In this case device A and device B both use the default MTU for the Attribute Protocol Response.
5. Once the MTU Exchange Request has been sent, the initiating device shall not send an Attribute Protocol Indication or Notification until after the MTU Exchange Response has been received. Note: This stops the risk of a cross-over condition where the MTU size is unknown for the Indication or Notification.

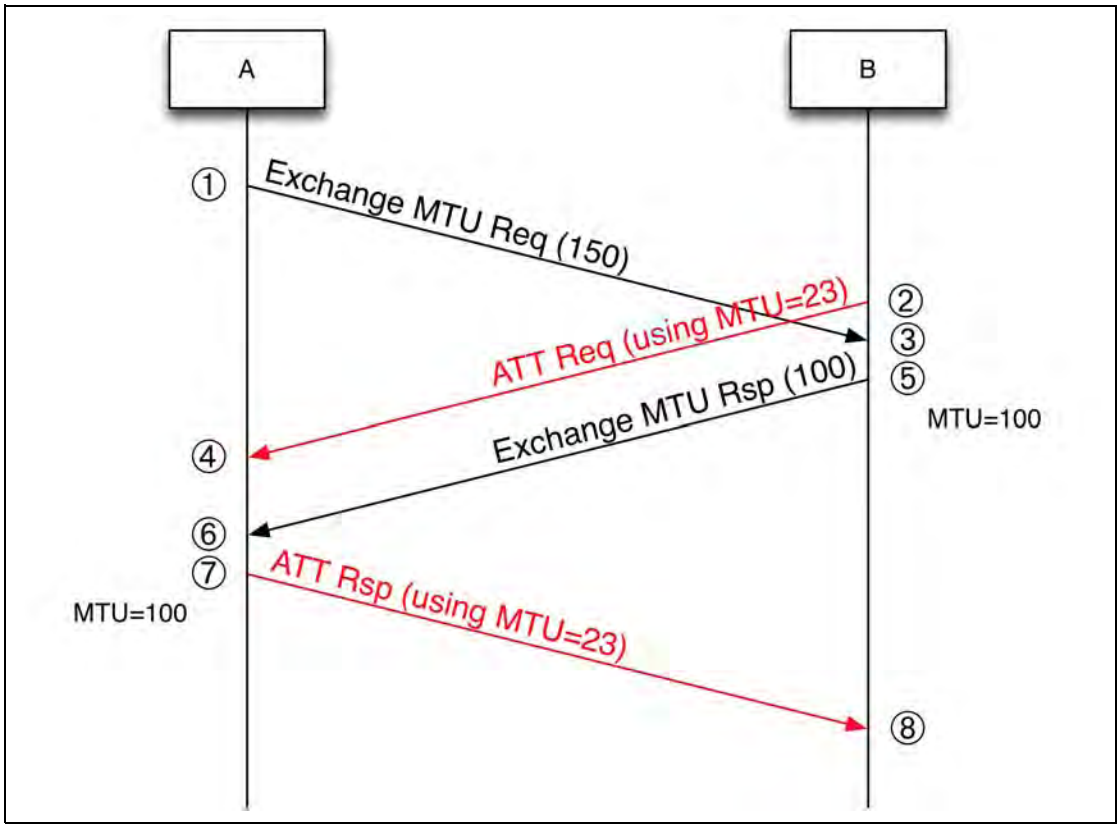



*Figure 3.1: MTU request and response exchange*

### 3.4.3 Find Information

#### *3.4.3.1 Find Information Request*

The *Find Information Request* is used to obtain the mapping of attribute handles with their associated types. This allows a client to discover the list of attributes and their types on a server.

| Parameter | Size (octets) | Description |
|---|---|---|
| Attribute Opcode | 1 | 0x04 = Find Information Request |
| Starting Handle | 2 | First requested handle number |
| Ending Handle | 2 | Last requested handle number |

*Table 3.6: Format of Find Information Request*

Only attributes with attribute handles between and including the Starting Handle parameter and the Ending Handle parameter will be returned. To read all attributes, the Starting Handle parameter shall be set to 0x0001, and the Ending Handle parameter shall be set to 0xFFFF. The Starting Handle parameter shall be less than or equal to the Ending Handle parameter.

If one or more attributes will be returned, a *Find Information Response* PDU shall be sent.

If a server receives a *Find Information Request* with the Starting Handle parameter greater than the Ending Handle parameter or the Starting Handle parameter is 0x0000, an *Error Response* shall be sent with the «Invalid Handle» error code; the Attribute Handle In Error parameter shall be set to the Starting Handle parameter.

If no attributes will be returned, an *Error Response* shall be sent with the «Attribute Not Found» error code; the Attribute Handle In Error parameter shall be set to the Starting Handle parameter.

The server shall not respond to the *Find Information Request* with an *Error Response* with the «Insufficient Authentication», «Insufficient Authorization», «Insufficient Encryption Key Size» or «Application Error» error code.

### *3.4.3.2 Find Information Response*

The *Find Information Response* is sent in reply to a received *Find Information Request* and contains information about this server.

| Parameter | Size (octets) | Description |
|---|---|---|
| Attribute Opcode | 1 | 0x05 = Find Information Response |
| Format | 1 | The format of the information data. |
| Information Data | 4 to (ATT_MTU–2) | The information data whose format is determined by the Format field |

*Table 3.7: Format of Find Information Response*

The *Find Information Response* shall have complete handle-UUID pairs. Such pairs shall not be split across response packets; this also implies that a handle-UUID pair shall fit into a single response packet. The handle-UUID pairs shall be returned in ascending order of attribute handles.

The Format parameter can contain one of two possible values.

| Name | Format | Description |
|---|---|---|
| Handle(s) and 16-bit Bluetooth UUID(s) | 0x01 | A list of 1 or more handles with their 16-bit Bluetooth UUIDs |
| Handle(s) and 128-bit UUID(s) | 0x02 | A list of 1 or more handles with their 128-bit UUIDs |

*Table 3.8: Format field values*

The information data field is comprised of a list of data defined in the tables below depending on the value chosen for the format.

| Handle | 16-bit Bluetooth UUID |
|---|---|
| 2 octets | 2 octets |

*Table 3.9: Format 0x01 - handle and 16-bit Bluetooth UUIDs*

| Handle | 128-bit UUID |
|---|---|
| 2 octets | 16 octets |

*Table 3.10: Format 0x02 - handle and 128-bit UUIDs*

Note: If sequential attributes have differing UUID sizes, it may happen that a *Find Information Response* is not filled with the maximum possible amount of (handle, UUID) pairs. This is because it is not possible to include attributes with differing UUID sizes into a single response packet. In that case, the following attribute would have to be read using another *Find Information Request* with its starting handle updated.

### *3.4.3.3 Find By Type Value Request*

The *Find By Type Value Request* is used to obtain the handles of attributes that have a 16-bit UUID attribute type and attribute value.This allows the range of handles associated with a given attribute to be discovered when the attribute type determines the grouping of a set of attributes. Note: Generic Attribute Profile defines grouping of attributes by attribute type.

| Parameter | Size (octets) | Description |
|---|---|---|
| Attribute Opcode | 1 | 0x06 = Find By Type Value Request |
| Starting Handle | 2 | First requested handle number |
| Ending Handle | 2 | Last requested handle number |
| Attribute Type | 2 | 2 octet UUID to find |
| Attribute Value | 0 to (ATT_MTU–7) | Attribute value to find |

*Table 3.11: Format of Find By Type Value Request*

Only attributes with attribute handles between and including the Starting Handle parameter and the Ending Handle parameter that match the requested attribute type and the attribute value that have sufficient permissions to allow reading will be returned. To read all attributes, the Starting Handle parameter shall be set to 0x0001, and the Ending Handle parameter shall be set to 0xFFFF.

If one or more handles will be returned, a *Find By Type Value Response* PDU shall be sent.

Note: Attribute values will be compared in terms of length and binary representation.

Note: It is not possible to use this request on an attribute that has a value longer than (ATT_MTU–7).

If a server receives a *Find By Type Value Request* with the Starting Handle parameter greater than the Ending Handle parameter or the Starting Handle parameter is 0x0000, an *Error Response* shall be sent with the «Invalid Handle» error code. The Attribute Handle In Error parameter shall be set to the Starting Handle parameter.

If no attributes will be returned, an *Error Response* shall be sent by the server with the error code «Attribute Not Found». The Attribute Handle In Error parameter shall be set to the starting handle.

The server shall not respond to the *Find By Type Value Request* with an *Error Response* with the «Insufficient Authentication», «Insufficient Authorization», «Insufficient Encryption Key Size», «Insufficient Encryption» or «Application Error» error code.

### *3.4.3.4 Find By Type Value Response*

The *Find By Type Value Response* is sent in reply to a received *Find By Type Value Request* and contains information about this server.

| Parameter | Size (octets) | Description |
|---|---|---|
| Attribute Opcode | 1 | 0x07 = Find By Type Value Response |
| Handles Information List | to (ATT_MTU–1) | A list of 1 or more Handle Informations. |

*Table 3.12: Format of Find By Type Value Response*

The Handles Information List field is a list of one or more Handle Informations.The Handles Information field is an attribute handle range as defined in Table 3.13.

| Found Attribute Handle | Group End Handle |
|---|---|
| 2 octets | 2 octets |

*Table 3.13: Format of the Handles Information*

The *Find By Type Value Response* shall contain one or more complete Handles Information. Such Handles Information shall not be split across response packets. The Handles Information List is ordered sequentially based on the found attribute handles.

For each handle that matches the attribute type and attribute value in the *Find By Type Value Request* a Handles Information shall be returned. The Found Attribute Handle shall be set to the handle of the attribute that has the exact attribute type and attribute value from the *Find By Type Value Request*. If the

attribute type in the *Find By Type Value Request* is a grouping attribute as defined by a higher layer specification, the Group End Handle shall be defined by that higher layer specification. If the attribute type in the *Find By Type Value Request* is not a grouping attribute as defined by a higher layer specification, the Group End Handle shall be equal to the Found Attribute Handle. If no other attributes with the same attribute type exist after the Found Attribute Handle, the End Found Handle shall be set to 0xFFFF.

Note: The Group End Handle may be greater than the Ending Handle in the *Find By Type Value Request*.

If a server receives a *Find By Type Value Request*, the server shall respond with the *Find By Type Value Response* containing as many handles for attributes that match the requested attribute type and attribute value that exist in the server that will fit into the maximum PDU size of (ATT_MTU–1).

### 3.4.4 Reading Attributes

#### *3.4.4.1 Read By Type Request*

The *Read By Type Request* is used to obtain the values of attributes where the attribute type is known but the handle is not known.

| Parameter | Size (octets) | Description |
|---|---|---|
| Attribute Opcode | 1 | 0x08 = Read By Type Request |
| Starting Handle | 2 | First requested handle number |
| Ending Handle | 2 | Last requested handle number |
| Attribute Type | 2 or 16 | 2 or 16 octet UUID |

*Table 3.14: Format of Read By Type Request*

Only the attributes with attribute handles between and including the Starting Handle and the Ending Handle with the attribute type that is the same as the Attribute Type given will be returned. To search through all attributes, the starting handle shall be set to 0x0001 and the ending handle shall be set to 0xFFFF.

Note: All attribute types are effectively compared as 128-bit UUIDs, even if a 16-bit UUID is provided in this request or defined for an attribute. See Section 3.2.1.

The starting handle shall be less than or equal to the ending handle. If a server receives a *Read By Type Request* with the Starting Handle parameter greater than the Ending Handle parameter or the Starting Handle parameter is 0x0000, an *Error Response* shall be sent with the «Invalid Handle» error code; The Attribute Handle In Error parameter shall be set to the Starting Handle parameter.

If no attribute with the given type exists within the handle range, then no attribute handle and value will be returned, and an *Error Response* shall be sent with the error code «Attribute Not Found». The Attribute Handle In Error parameter shall be set to the starting handle.

The attributes returned shall be the attributes with the lowest handles within the handle range. These are known as the requested attributes.

If the attributes with the requested type within the handle range have attribute values that have the same length, then these attributes can all be read in a single request.

The attribute server shall include as many attributes as possible in the response in order to minimize the number of PDUs required to read attributes of the same type.

Note: If the attributes with the requested type within the handle range have attribute values with different lengths, then multiple *Read By Type Request*s must be made.

When multiple attributes match, then the rules below shall be applied to each in turn.

- Only attributes that can be read shall be returned in a *Read By Type Response*.
- If an attribute in the set of requested attributes would cause an *Error Response* then this attribute cannot be included in a *Read By Type Response* and the attributes before this attribute shall be returned.
- If the first attribute in the set of requested attributes would cause an *Error Response* then no other attributes in the requested attributes can be considered.

The server shall respond with a *Read By Type Response* if the requested attributes have sufficient permissions to allow reading.

If the client has insufficient authorization to read the requested attribute then an *Error Response* shall be sent with the error code «Insufficient Authorization». The Attribute Handle In Error parameter shall be set to the handle of the attribute causing the error.

If the client has insufficient security to read the requested attribute then an *Error Response* shall be sent with the error code «Insufficient Authentication». The Attribute Handle In Error parameter shall be set to the handle of the attribute causing the error.

If the client has an insufficient encryption key size to read the requested attribute then an *Error Response* shall be sent with the error code «Insufficient Encryption Key Size». The Attribute Handle In Error parameter shall be set to the handle of the attribute causing the error.

If the client has not enabled encryption, and encryption is required to read the requested attribute, then an *Error Response* shall be sent with the error code «Insufficient Encryption». The Attribute Handle In Error parameter shall be set to the handle of the attribute causing the error.

If the requested attribute's value cannot be read due to permissions then an *Error Response* shall be sent with the error code «Read Not Permitted». The Attribute Handle In Error parameter shall be set to the handle of the attribute causing the error.

Note: if there are multiple attributes with the requested type within the handle range, and the client would like to get the next attribute with the requested type, it would have to issue another *Read By Type Request* with its starting handle updated. The client can be sure there are no more such attributes remaining once it gets an *Error Response* with the error code «Attribute Not Found».

### *3.4.4.2 Read By Type Response*

The *Read By Type Response* is sent in reply to a received *Read By Type Request* and contains the handles and values of the attributes that have been read.

| Parameter | Size (octets) | Description |
|---|---|---|
| Attribute Opcode | 1 | 0x09 = Read By Type Response |
| Length | 1 | The size of each attribute handle-value pair |
| Attribute Data List | 2 to (ATT_MTU– 2) | A list of Attribute Data. |

*Table 3.15: Format of Read By Type Response*

The *Read By Type Response* shall contain complete handle-value pairs. Such pairs shall not be split across response packets. The handle-value pairs shall be returned sequentially based on the attribute handle.

The Length parameter shall be set to the size of one attribute handle-value pair.

The maximum length of an attribute handle-value pair is 255 octets, bounded by the Length parameter that is one octet. Therefore, the maximum length of an attribute value returned in this response is (Length – 2) = 253 octets.

The attribute handle-value pairs shall be set to the value of the attributes identified by the attribute type within the handle range within the request. If the attribute value is longer than (ATT_MTU - 4) or 253 octets, whichever is smaller, then the first (ATT_MTU- 4) or 253 octets shall be included in this response.

Note: the *Read Blob Request* would be used to read the remaining octets of a long attribute value.

The Attribute Data field is comprised of a list of attribute handle and value pairs as defined in Table 3.16.

| Attribute Handle | Attribute Value |
|---|---|
| 2 octets | (Length – 2) octets |

*Table 3.16: Format of the Attribute Data*

### *3.4.4.3 Read Request*

The *Read Request* is used to request the server to read the value of an attribute and return its value in a *Read Response*.

| Parameter | Size (octets) | Description |
|---|---|---|
| Attribute Opcode | 1 | 0x0A = Read Request |
| Attribute Handle | 2 | The handle of the attribute to be read |

*Table 3.17: Format of Read Request*

The attribute handle parameter shall be set to a valid handle.

The server shall respond with a *Read Response* if the handle is valid and the attribute has sufficient permissions to allow reading.

If the client has insufficient authorization to read the requested attribute then an *Error Response* shall be sent with the error code «Insufficient Authorization».

If the client has insufficient security to read the requested attribute then an *Error Response* shall be sent with the error code «Insufficient Authentication».

If the client has an insufficient encryption key size to read the requested attribute then an *Error Response* shall be sent with the error code «Insufficient Encryption Key Size».

If the client has not enabled encryption, and encryption is required to read the requested attribute, then an *Error Response* shall be sent with the error code «Insufficient Encryption».

If the handle is invalid, then an *Error Response* shall be sent with the error code «Invalid Handle».

If the attribute value cannot be read due to permissions then an *Error Response* shall be sent with the error code «Read Not Permitted».

### *3.4.4.4 Read Response*

The read response is sent in reply to a received *Read Request* and contains the value of the attribute that has been read.

| Parameter | Size (octets) | Description |
|---|---|---|
| Attribute Opcode | 1 | 0x0B = Read Response |
| Attribute Value | 0 to (ATT_MTU–1) | The value of the attribute with the handle given |

*Table 3.18: Format of Read Response*

The attribute value shall be set to the value of the attribute identified by the attribute handle in the request. If the attribute value is longer than (ATT_MTU–1) then the first (ATT_MTU–1) octets shall be included in this response.

Note: the *Read Blob Request* would be used to read the remaining octets of a long attribute value.

### *3.4.4.5 Read Blob Request*

The *Read Blob Request* is used to request the server to read part of the value of an attribute at a given offset and return a specific part of the value in a *Read Blob Response*.

| Parameter | Size (octets) | Description |
|---|---|---|
| Attribute Opcode | 1 | 0x0C = Read Blob Request |
| Attribute Handle | 2 | The handle of the attribute to be read |
| Value Offset | 2 | The offset of the first octet to be read |

*Table 3.19: Format of Read Blob Request*

The attribute handle parameter shall be set to a valid handle.

The value offset parameter is based from zero; the first value octet has an offset of zero, the second octet has a value offset of one, etc.

The server shall respond with a *Read Blob Response* if the handle is valid and the attribute and value offset is not greater than the length of the attribute value and has sufficient permissions to allow reading.

If the client has insufficient authorization to read the requested attribute then an *Error Response* shall be sent with the error code «Insufficient Authorization».

If the client has insufficient security to read the requested attribute then an *Error Response* shall be sent with the error code «Insufficient Authentication».

If the client has an insufficient encryption key size to read the requested attribute then an *Error Response* shall be sent with the error code «Insufficient Encryption Key Size».

If the client has not enabled encryption, and encryption is required to read the requested attribute, then an *Error Response* shall be sent with the error code «Insufficient Encryption».

If the handle is invalid, then an *Error Response* shall be sent with the error code «Invalid Handle».

If the attribute value cannot be read due to permissions then an *Error Response* shall be sent with the error code «Read Not Permitted».

If the value offset of the *Read Blob Request* is greater than the length of the attribute value, an *Error Response* shall be sent with the error code «Invalid Offset».

*Error Response*

If the attribute value has a fixed length that is less than or equal to (ATT_MTU – 3) octets in length, then an *Error Response* can be sent with the error code «Attribute Not Long.»

If the value offset of the *Read Blob Request* is equal to the length of the attribute value, then the length of the part attribute value in the response shall be zero.

Note: if the attribute is longer than (ATT_MTU–1) octets, the Read Blob Request is the only way to read the additional octets of a long attribute. The first (ATT_MTU–1) octets may be read using a *Read Request*, an *Handle Value Notification* or an *Handle Value Indication*.

Note: Long attributes may or may not have their length specified by a higher layer specification. If the long attribute has a variable length, the only way to get to the end of it is to read it part by part until the value in the *Read Blob Response* has a length shorter than (ATT_MTU–1) or an *Error Response* with the error code «Invalid Offset».

Note: the value of a Long Attribute may change between one Read Blob Request and the next Read Blob Request. A higher layer specification should be aware of this and define appropriate behavior.

### *3.4.4.6 Read Blob Response*

The *Read Blob Response* is sent in reply to a received *Read Blob Request* and contains part of the value of the attribute that has been read.

| Parameter | Size (octets) | Description |
|---|---|---|
| Attribute Opcode | 1 | 0x0D = Read Blob Response |

*Table 3.20: Format of Read Blob Response*

| Parameter | Size (octets) | Description |
|---|---|---|
| Part Attribute Value | 0 to (ATT_MTU–1) | Part of the value of the attribute with the handle given |

*Table 3.20: Format of Read Blob Response*

The part attribute value shall be set to part of the value of the attribute identified by the attribute handle and the value offset in the request. If the value offset is equal to the length of the attribute value, then the length of the part attribute value shall be zero. If the attribute value is longer than (Value Offset + ATT_MTU–1) then (ATT_MTU–1) octets from Value Offset shall be included in this response.

### *3.4.4.7 Read Multiple Request*

The *Read Multiple Request* is used to request the server to read two or more values of a set of attributes and return their values in a *Read Multiple Response*. Only values that have a known fixed size can be read, with the exception of the last value that can have a variable length. The knowledge of whether attributes have a known fixed size is defined in a higher layer specification.

| Parameter | Size (octets) | Description |
|---|---|---|
| Attribute Opcode | 1 | 0x0E = Read Multiple Request |
| Set Of Handles | 4 to (ATT_MTU–1) | A set of two or more attribute handles. |

*Table 3.21: Format of Read Multiple Request*

The attribute handles in the Set Of Handles parameter shall be valid handles.

The server shall respond with a *Read Multiple Response* if all the handles are valid and all attributes have sufficient permissions to allow reading.

Note: The attribute values for the attributes in the Set Of Handles parameters do not have to all be the same size.

Note: The attribute handles in the Set Of Handles parameter do not have to be in attribute handle order; they are in the order that the values are required in the response.

If the client has insufficient authorization to read any of the attributes then an *Error Response* shall be sent with the error code «Insufficient Authorization».

If the client has insufficient security to read any of the attributes then an *Error Response* shall be sent with the error code «Insufficient Authentication».

If the client has an insufficient encryption key size to read any of the attributes then an *Error Response* shall be sent with the error code «Insufficient Encryption Key Size».

If the client has not enabled encryption, and encryption is required to read the requested attribute, then an *Error Response* shall be sent with the error code «Insufficient Encryption».

If any of the handles are invalid, then an *Error Response* shall be sent with the error code «Invalid Handle».

If any of the attribute values cannot be read due to permissions then an *Error Response* shall be sent with the error code «Read Not Permitted».

If an *Error Response* is sent, the Attribute Handle In Error parameter shall be set to the handle of the first attribute causing the error.

### *3.4.4.8 Read Multiple Response*

The read response is sent in reply to a received *Read Multiple Request* and contains the values of the attributes that have been read.

| Parameter | Size (octets) | Description |
|---|---|---|
| Attribute Opcode | 1 | 0x0F = Read Multiple Response |
| Set Of Values | 0 to (ATT_MTU–1) | A set of two or more values. |

*Table 3.22: Format of Read Multiple Response*

The Set Of Values parameter shall be a concatenation of attribute values for each of the attribute handles in the request in the order that they were requested. If the Set Of Values parameter is longer than (ATT_MTU–1) then only the first (ATT_MTU–1) octets shall be included in this response.

Note: a client should not use this request for attributes when the Set Of Values parameter could be (ATT_MTU–1) as it will not be possible to determine if the last attribute value is complete, or if it overflowed.

### *3.4.4.9 Read by Group Type Request*

The *Read By Group Type Request* is used to obtain the values of attributes where the attribute type is known, the type of a grouping attribute as defined by a higher layer specification, but the handle is not known.

| Parameter | Size (octets) | Description |
|---|---|---|
| Attribute Opcode | 1 | 0x10 = Read By Group Type Request |
| Starting Handle | 2 | First requested handle number |

*Table 3.23: Format of Read By Group Type Request*

| Parameter | Size (octets) | Description |
|---|---|---|
| Ending Handle | 2 | Last requested handle number |
| Attribute Group Type | 2 or 16 | 2 or 16 octet UUID |

*Table 3.23: Format of Read By Group Type Request*

Only the attributes with attribute handles between and including the Starting Handle and the Ending Handle with the attribute type that is the same as the Attribute Group Type given will be returned. To search through all attributes, the starting handle shall be set to 0x0001 and the ending handle shall be set to 0xFFFF.

Note: All attribute types are effectively compared as 128-bit UUIDs, even if a 16-bit UUID is provided in this request or defined for an attribute. See Section 3.2.1.

The starting handle shall be less than or equal to the ending handle. If a server receives a *Read By Group Type Request* with the Starting Handle parameter greater than the Ending Handle parameter or the Starting Handle parameter is 0x0000, an *Error Response* shall be sent with the «Invalid Handle» error code; The Attribute Handle In Error parameter shall be set to the Starting Handle parameter.

If the Attribute Group Type is not a supported grouping attribute as defined by a higher layer specification then an *Error Response* shall be sent with the error code «Unsupported Group Type». The Attribute Handle In Error parameter shall be set to the Starting Handle.

If no attribute with the given type exists within the handle range, then no attribute handle and value will be returned, and an *Error Response* shall be sent with the error code «Attribute Not Found». The Attribute Handle In Error parameter shall be set to the starting handle.

The attributes returned shall be the attributes with the lowest handles within the handle range. These are known as the requested attributes.

If the attributes with the requested type within the handle range have attribute values that have the same length, then these attributes can all be read in a single request.

The attribute server shall include as many attributes as possible in the response in order to minimize the number of PDUs required to read attributes of the same type.

Note: If the attributes with the requested type within the handle range have attribute values with different lengths, then multiple *Read By Group Type Request* s must be made.

When multiple attributes match, then the rules below shall be applied to each in turn.

- Only attributes that can be read shall be returned in a *Read By Group Type Response*.
- If an attribute in the set of requested attributes would cause an *Error Response* then this attribute cannot be included in a *Read By Group Type Response* and the attributes before this attribute shall be returned.
- If the first attribute in the set of requested attributes would cause an *Error Response* then no other attributes in the requested attributes can be considered.

The server shall respond with a *Read By Group Type Response* if the requested attributes have sufficient permissions to allow reading.

If the client has insufficient authorization to read the requested attribute then an *Error Response* shall be sent with the error code «Insufficient Authorization». The Attribute Handle In Error parameter shall be set to the handle of the attribute causing the error.

If the client has insufficient security to read the requested attribute then an *Error Response* shall be sent with the error code «Insufficient Authentication». The Attribute Handle In Error parameter shall be set to the handle of the attribute causing the error.

If the client has an insufficient encryption key size to read the requested attribute then an *Error Response* shall be sent with the error code «Insufficient Encryption Key Size». The Attribute Handle In Error parameter shall be set to the handle of the attribute causing the error.

If the client has not enabled encryption, and encryption is required to read the requested attribute, then an *Error Response* shall be sent with the error code «Insufficient Encryption». The Attribute Handle In Error parameter shall be set to the handle of the attribute causing the error.

If the requested attribute's value cannot be read due to permissions then an *Error Response* shall be sent with the error code «Read Not Permitted». The Attribute Handle In Error parameter shall be set to the handle of the attribute causing the error.

Note: if there are multiple attributes with the requested type within the handle range, and the client would like to get the next attribute with the requested type, it would have to issue another *Read By Group Type Request* with its starting handle updated. The client can be sure there are no more such attributes remaining once it gets an *Error Response* with the error code «Attribute Not Found».

#### *3.4.4.10 Read by Group Type Response*

The *Read By Group Type Response* is sent in reply to a received *Read By Group Type Request* and contains the handles and values of the attributes that have been read.

| Parameter | Size (octets) | Description |
|---|---|---|
| Attribute Opcode | 1 | 0x11 = Read By Group Type Response |
| Length | 1 | The size of each Attribute Data |
| Attribute Data List | 2 to (ATT_MTU– 2) | A list of Attribute Data. |

*Table 3.24: Format of Read By Group Type Response*

The *Read By Group Type Response* shall contain complete Attribute Data. An Attribute Data shall not be split across response packets. The Attribute Data List is ordered sequentially based on the attribute handles

The Length parameter shall be set to the size of the one Attribute Data.

The maximum length of an Attribute Data is 255 octets, bounded by the Length parameter that is one octet. Therefore, the maximum length of an attribute value returned in this response is (Length – 4) = 251 octets.

The Attribute Data List shall be set to the value of the attributes identified by the attribute type within the handle range within the request. If the attribute value is longer than (ATT_MTU - 6) or 251 octets, whichever is smaller, then the first (ATT_MTU - 6) or 251 octets shall be included in this response.

Note: the *Read Blob Request* would be used to read the remaining octets of a long attribute value.

The Attribute Data List is comprised of a list of Attribute Data as defined in Table 3.25.

| Attribute Handle | End Group Handle | Attribute Value |
|---|---|---|
| 2 octets | 2 octets | (Length - 4) octets |

*Table 3.25: Format of the Attribute Data*

### 3.4.5 Writing Attributes

#### *3.4.5.1 Write Request*

The *Write Request* is used to request the server to write the value of an attribute and acknowledge that this has been achieved in a *Write Response*.

| Parameter | Size (octets) | Description |
|---|---|---|
| Attribute Opcode | 1 | 0x12 = Write Request |
| Attribute Handle | 2 | The handle of the attribute to be written |
| Attribute Value | 0 to (ATT_MTU–3) | The value to be written to the attribute |

*Table 3.26: Format of Write Request*

The Attribute Handle shall be set to a valid handle.

The Attribute Value shall be set to the new value of the attribute.

If the attribute value has a variable length, then the attribute value shall be truncated or lengthened to match the length of the Attribute Value parameter.

Note: If an attribute value has a variable length and if the Attribute Value parameter is of zero length, the attribute value will be fully truncated.

If the attribute value has a fixed length and the Attribute Value parameter length is less than or equal to the length of the attribute value, the octets up the attribute value parameter length shall be written; all other octets in this attribute value shall be unchanged.

The server shall respond with a *Write Response* if the handle is valid, the attribute has sufficient permissions to allow writing, and the attribute value has a valid size and format, and it is successful in writing the attribute.

If the attribute value has a variable length and the Attribute Value parameter length exceeds the maximum valid length of the attribute value then the server shall respond with an *Error Response* with the error code «Invalid Attribute Value Length».

If the attribute value has a fixed length and the requested attribute value parameter length is greater than the length of the attribute value then the server shall respond with an *Error Response* with the error code «Invalid Attribute Value Length».

If the client has insufficient authorization to write the requested attribute then an *Error Response* shall be sent with the error code «Insufficient Authorization».

If the client has insufficient security to write the requested attribute then an *Error Response* shall be sent with the error code «Insufficient Authentication».

If the client has an insufficient encryption key size to write the requested attribute then an *Error Response* shall be sent with the error code «Insufficient Encryption Key Size».

If the client has not enabled encryption, and encryption is required to write the requested attribute, then an *Error Response* shall be sent with the error code «Insufficient Encryption».

If the handle is invalid, then an *Error Response* shall be sent with the error code «Invalid Handle».

If the attribute value cannot be written due to permissions then an *Error Response* shall be sent with the error code «Write Not Permitted».

If the attribute value cannot be written due to an application error then an *Error Response* shall be sent with an error code defined by a higher layer specification.

#### *3.4.5.2 Write Response*

The *Write Response* is sent in reply to a valid *Write Request* and acknowledges that the attribute has been successfully written.

| Parameter | Size (octets) | Description |
|---|---|---|
| Attribute Opcode | 1 | 0x13 = Write Response |

*Table 3.27: Format of Write Response*

The *Write Response* shall be sent after the attribute value is written.

#### *3.4.5.3 Write Command*

The *Write Command* is used to request the server to write the value of an attribute, typically into a control-point attribute.

| Parameter | Size (octets) | Description |
|---|---|---|
| Attribute Opcode | 1 | 0x52 = Write Command |
| Attribute Handle | 2 | The handle of the attribute to be set |
| Attribute Value | 0 to (ATT_MTU–3) | The value of be written to the attribute |

*Table 3.28: Format of Write Command*

The attribute handle parameter shall be set to a valid handle.

The attribute value parameter shall be set to the new value of the attribute.

If the attribute value has a variable length, then the attribute value shall be truncated or lengthened to match the length of the attribute value parameter.

Note: If an attribute value has a variable length and if the attribute value parameter is of zero length, the attribute value will be fully truncated.

If the attribute value has a fixed length and the attribute value parameter length is less than or equal to the length of the attribute value, the octets up to the attribute value parameter length shall be written; all other octets in this attribute value shall be unchanged.

If the attribute value has a variable length and the attribute value parameter length exceeds the maximum valid length of the attribute value then the server shall ignore the command.

If the attribute value has a fixed length and the requested attribute value parameter length is greater than the length of the attribute value then the server shall ignore the command.

No *Error Response* or *Write Response* shall be sent in response to this command. If the server cannot write this attribute for any reason the command shall be ignored.

### *3.4.5.4 Signed Write Command*

The *Signed Write Command* is used to request the server to write the value of an attribute with an authentication signature, typically into a control-point attribute.

| Parameter | Size (Octets) | Description |
|---|---|---|
| Attribute Opcode | 1 | 0xD2 = Signed Write Command |
| Attribute Handle | 2 | The handle of the attribute to be set |
| Attribute Value | 0 to (ATT_MTU - 13) | The value to be written to the attribute |
| Authentication Signature | 12 | Authentication signature for the Attribute Upload, Attribute Handle and Attribute Value Parameters |

*Table 3.29: Format of Signed Write Command*

The attribute handle parameter shall be set to a valid handle.

The attribute value parameter shall be set to the new value of the attribute.

The attribute signature shall be calculated as defined in Section 3.3.1.

If the attribute value has a variable length, then the attribute value shall be truncated or lengthened to match the length of the attribute value parameter.

Note: If an attribute value has a variable length and if the attribute value parameter is of zero length, the attribute value will be fully truncated.

If the attribute value has a fixed length and the attribute value parameter length is less than or equal to the length of the attribute value, the octets up to the attribute value parameter length shall be written; all other octets in this attribute value shall be unchanged.

If the attribute value has a variable length and the attribute value parameter length exceeds the maximum valid length of the attribute value then the server shall ignore the command.

If the attribute value has a fixed length and the requested attribute value parameter length is greater than the length of the attribute value then the server shall ignore the command.

If the authentication signature verification fails, then the server shall ignore the command.

No *Error Response* or *Write Response* shall be sent in response to this command. If the server cannot write this attribute for any reason the command shall be ignored.

### 3.4.6 Queued Writes

The purpose of queued writes is to queue up writes of values of multiple attributes in a first-in first-out queue and then execute the write on all of them in a single atomic operation.

#### *3.4.6.1 Prepare Write Request*

The *Prepare Write Request* is used to request the server to prepare to write the value of an attribute. The server will respond to this request with a *Prepare Write Response*, so that the client can verify that the value was received correctly.

A client may send more than one *Prepare Write Request* to a server, which will queue and send a response for each handle value pair.

A server may limit the number of prepare write requests that it can accept. A higher layer specification should define this limit.

After a *Prepare Write Request* has been issued, and the response received, any other attribute command or request can be issued from the same client to the same server.

Each client's queued values are separate; the execution of one queue shall not affect the preparation or execution of any other client's queued values.

Any actions on attributes that exist in the prepare queue shall proceed as if the prepare queue did not exist, and the prepare queue shall be unaffected by these actions. A subsequent execute write will write the values in the prepare queue even if the value of the attribute has changed since the prepare writes were started.

The attribute protocol makes no determination on the validity of the Part Attribute Value or the Value Offset. A higher layer specification determines the meaning of the data.

Each *Prepare Write Request* will be queued even if the attribute handle is the same as a previous *Prepare Write Request*. These will then be executed in the order received, causing multiple writes for this attribute to occur.

If the link is lost while a number of prepared write requests have been queued, the queue will be cleared and no writes will be executed.

| Parameter | Size (octets) | Description |
|---|---|---|
| Attribute Opcode | 1 | 0x16 = Prepare Write Request |
| Attribute Handle | 2 | The handle of the attribute to be written |
| Value Offset | 2 | The offset of the first octet to be written |
| Part Attribute Value | 0 to (ATT_MTU–5 | The value of the attribute to be written |

*Table 3.30: Format of Prepare Write Request*

The Attribute Handle parameter shall be set to a valid handle.

The Value Offset parameter shall be set to the offset of the first octet where the Part Attribute Value parameter is to be written within the attribute value. The Value Offset parameter is based from zero; the first octet has an offset of zero, the second octet has an offset of one, etc.

The server shall respond with a *Prepare Write Response* if the handle is valid, the attribute has sufficient permissions to allow writing at this time, and the prepare queue has sufficient space.

Note: The Attribute Value validation is done when an Execute Write Request is received. Hence, any Invalid Offset or Invalid Attribute Value Length errors are generated when an Execute Write Request is received.

If the client has insufficient authorization to write the requested attribute then an *Error Response* shall be sent with the error code «Insufficient Authorization».

If the client has insufficient security to write the requested attribute then an *Error Response* shall be sent with the error code «Insufficient Authentication».

If the client has an insufficient encryption key size to write the requested attribute then an *Error Response* shall be sent with the error code «Insufficient Encryption Key Size».

If the client has not enabled encryption, and encryption is required to write the requested attribute, then an *Error Response* shall be sent with the error code «Insufficient Encryption».

If the server does not have sufficient space to queue this request then an *Error Response* shall be sent with the error code «Prepare Queue Full».

If the handle is invalid, then an *Error Response* shall be sent with the error code «Invalid Handle».

If the attribute value cannot be written then an *Error Response* shall be sent with the error code «Write Not Permitted».

The server shall not change the value of the attribute until an *Execute Write Request* is received.

If a *Prepare Write Request* was invalid, and therefore an *Error Response* has been issued, then this prepared write will be considered to have not been received. All existing prepared writes in the prepare queue shall not be affected by this invalid request.

#### *3.4.6.2 Prepare Write Response*

The *Prepare Write Response* is sent in response to a received *Prepare Write Request* and acknowledges that the value has been successfully received and placed in the prepare write queue.

| Parameter | Size (octets) | Description |
|---|---|---|
| Attribute Opcode | 1 | 0x17 = Prepare Write Response |
| Attribute Handle | 2 | The handle of the attribute to be written |
| Value Offset | 2 | The offset of the first octet to be written |
| +Part Attribute Value | 0 to (ATT_MTU–5)) | The value of the attribute to be written |

*Table 3.31: Format of Prepare Write Response*

The attribute handle shall be set to the same value as in the corresponding *Prepare Write Request*.

The value offset and part attribute value shall be set to the same values as in the corresponding *Prepare Write Request*.

### 3.4.6.3 Execute Write Request

The *Execute Write Request* is used to request the server to write or cancel the write of all the prepared values currently held in the prepare queue from this client. This request shall be handled by the server as an atomic operation.

| Parameter | Size (octets) | Description |
|---|---|---|
| Attribute Opcode | 1 | 0x18 = Execute Write Request |
| Flags | 1 | 0x00 – Cancel all prepared writes<br>0x01 – Immediately write all pending prepared values |

*Table 3.32: Format of Execute Write Request*

When the flags parameter is set to 0x01, values that were queued by the previous prepare write requests shall be written in the order they were received in the corresponding *Prepare Write Request*. The queue shall then be cleared, and an *Execute Write Response* shall be sent.

When the flags parameter is set to 0x00 all pending prepare write values shall be discarded for this client. The queue shall then be cleared, and an *Execute Write Response* shall be sent.

If the prepared Attribute Value exceeds the maximum valid length of the attribute value then all pending prepare write values shall be discarded for this client, the queue shall then be cleared, and an *Error Response* shall be sent with the error code «Invalid Attribute Value Length».

If the prepared Attribute Value exceeds the maximum valid length of the attribute value then all pending prepare write values shall be discarded for this client, the queue shall then be cleared, and an *Error Response* shall be sent with the error code «Invalid Attribute Value Length».

If the prepare Value Offset is greater than the current length of the attribute value then all pending prepare write values shall be discarded for this client, the queue shall be cleared and then an *Error Response* shall be sent with the «Invalid Offset».

If the prepare write requests cannot be written, due to an application error, the queue shall be cleared and then an *Error Response* shall be sent with a higher layer specification defined error code. The Attribute Handle In Error parameter shall be set to the attribute handle of the attribute from the prepare queue that caused this application error. The state of the attributes that were to be written from the prepare queue is not defined in this case.

#### *3.4.6.4 Execute Write Response*

The *Execute Write Response* is sent in response to a received *Execute Write Request*.

| Parameter | Size | Description |
|---|---|---|
| Attribute Upload | 1 | 0x19 - Execute Write Response |

*Table 3.33: Format of Execute Write Response[*

The *Execute Write Response* shall be sent after the attributes are written. In case an action is taken in response to the write, an indication may be used once the action is complete.

### 3.4.7 Server Initiated

#### *3.4.7.1 Handle Value Notification*

A server can send a notification of an attribute's value at any time.

| Parameter | Size (octets) | Description |
|---|---|---|
| Attribute Opcode | 1 | 0x1B = Handle Value Notification |
| Attribute Handle | 2 | The handle of the attribute |
| Attribute Value | 0 to (ATT_MTU–3) | The current value of the attribute |

*Table 3.34: Format of Handle Value Notification*

The attribute handle shall be set to a valid handle.

The attribute value shall be set to the current value of attribute identified by the attribute handle.

If the attribute value is longer than (ATT_MTU–3) octets, then only the first (ATT_MTU–3) octets of this attributes value can be sent in a notification.

Note: for a client to get a long attribute, it would have to use the *Read Blob Request* following the receipt of this notification.

If the attribute handle or the attribute value is invalid, then this notification shall be ignored upon reception.

#### *3.4.7.2 Handle Value Indication*

A server can send an indication of an attribute's value.

The attribute handle shall be set to a valid handle.

| Parameter | Size (octets) | Description |
|---|---|---|
| Attribute Opcode | 1 | 0x1D = Handle Value Indication |
| Attribute Handle | 2 | The handle of the attribute |
| Attribute Value | 0 to (ATT_MTU–3) | The current value of the attribute |

*Table 3.35: Format of Handle Value Indication*

The attribute value shall be set to the current value of attribute identified by the attribute handle.

If the attribute value is longer than (ATT_MTU–3) octets, then only the first (ATT_MTU - 3) octets of this attributes value can be sent in an indication.

Note: For a client to get a long attribute, it would have to use the *Read Blob Request* following the receipt of this indication.

The client shall send a *Handle Value Confirmation* in response to a *Handle Value Indication*. No further indications to this client shall occur until the confirmation has been received by the server.

If the attribute handle or the attribute value is invalid, the client shall send a handle value confirmation in response and shall discard the handle and value from the received indication.

#### *3.4.7.3 Handle Value Confirmation*

The *Handle Value Confirmation* is sent in response to a received *Handle Value Indication* and confirms that the client has received an indication of the given attribute.

| Parameter | Size (octets) | Description |
|---|---|---|
| Attribute Opcode | 1 | 0x1E = Handle Value Confirmation |

*Table 3.36: Format of Handle Value Confirmation*

### 3.4.8 Attribute Opcode Summary

Table 3.37 gives a summary of the attribute protocol PDUs. .

| Attribute PDU Name | Attribute Opcode | Parameters |
|---|---|---|
| **Error Response** | 0x01 | Request Opcode in Error,<br>Attribute Handle In Error,<br>Error Code |

*Table 3.37: Attribute Protocol Summary*

| Attribute PDU Name | Attribute Opcode | Parameters |
|---|---|---|
| **Exchange MTU Request** | 0x02 | Client Rx MTU |
| **Exchange MTU Response** | 0x03 | Server Rx MTU |
| **Find Information Request** | 0x04 | Starting Handle,<br>Ending Handle,<br>UUID |
| **Find Information Response** | 0x05 | Format,<br>Information Data |
| **Find By Type Value Request** | 0x06 | Starting Handle,<br>Ending Handle,<br>Attribute Type,<br>Attribute Value |
| **Find By Type Value Response** | 0x07 | Handles Information List |
| **Read By Type Request** | 0x08 | Starting Handle,<br>Ending Handle,<br>UUID |
| **Read By Type Response** | 0x09 | Length, Attribute Data List |
| **Read Request** | 0x0A | Attribute Handle |
| **Read Response** | 0x0B | Attribute Value |
| **Read Blob Request** | 0x0C | Attribute Handle,<br>Value Offset |
| **Read Blob Response** | 0x0D | Part Attribute Value |
| **Read Multiple Request** | 0x0E | Handle Set |
| **Read Multiple Response** | 0x0F | Value Set |
| **Read by Group Type Request** | 0x10 | Start Handle, Ending Handle, UUID |
| **Read by Group Type Response** | 0x11 | Length, Attribute Data List |
| **Write Request** | 0x12 | Attribute Handle,<br>Attribute Value |
| **Write Response** | 0x13 | - |

*Table 3.37: Attribute Protocol Summary*

| Attribute PDU Name | Attribute Opcode | Parameters |
|---|---|---|
| **Write Command** | 0x52 | Attribute Handle,<br>Attribute Value |
| **Prepare Write Request** | 0x16 | Attribute Handle,<br>Value Offset,<br>Part Attribute Value |
| **Prepare Write Response** | 0x17 | Attribute Handle,<br>Value Offset<br>Part Attribute Value |
| **Execute Write Request** | 0x18 | Flags |
| **Execute Write Response** | 0x19 | - |
| **Handle Value Notification** | 0x1B | Attribute Handle,<br>Attribute Value |
| **Handle Value Indication** | 0x1D | Attribute Handle,<br>Attribute Value |
| **Handle Value Confirmation** | 0x1E | |
| | | |
| **Signed Write Command** | 0xD2 | Attribute Handle,<br>Attribute Value,<br>Authentication Signature |
| | | |
| | | |
| | | |

*Table 3.37: Attribute Protocol Summary*

### 3.4.9 Attribute PDU Response Summary

Table 3.38 gives a summary of the Attribute PDU Method responses that are allowed. Each method indicates the method that should be sent as a successful response, and whether an Error Response can be sent in response instead. If an Error Response can be sent, then the table also indicates the Error Codes that are valid within this Error Response for the given method.

| Attribute PDU Method | Successful Response | Error Response Allowed | Error Response Error Codes |
|---|---|---|---|
| **Exchange MTU Request** | Exchange MTU Response | Yes | Request Not Supported |
| **Find Information Request** | Find Information Response | Yes | Invalid Handle,<br>Attribute Not Found |
| **Find By Type Value Request** | Find By Type Value Response | Yes | Invalid Handle,<br>Request Not Supported,<br>Attribute Not Found |
| **Read By Type Request** | Read By Type Response | Yes | Invalid Handle,<br>Request Not Supported,<br>Attribute Not Found,<br>Insufficient Authorization,<br>Insufficient Authentication,<br>Insufficient Encryption<br>Insufficient Encryption Key Size<br>Read Not Permitted,<br>Application Error |
| **Read Request** | Read Response | Yes | Invalid Handle,<br>Insufficient Authorization,<br>Insufficient Authentication,<br>Insufficient Encryption<br>Insufficient Encryption Key Size,<br>Read Not Permitted,<br>Application Error |

*Table 3.38: Attribute Request and Response Summary*

| Attribute PDU Method | Successful Response | Error Response Allowed | Error Response Error Codes |
|---|---|---|---|
| **Read Blob Request** | Read Blob Response | Yes | Invalid Handle<br>Request Not Supported,<br>Insufficient Authorization,<br>Insufficient Authentication,<br>Insufficient Encryption<br>Insufficient Encryption Key Size,<br>Read Not Permitted,<br>Invalid Offset,<br>Attribute Not Long,<br>Application Error |
| **Read Multiple Request** | Read Multiple Response | Yes | Invalid Handle,<br>Request Not Supported,<br>Insufficient Authorization,<br>Insufficient Authentication,<br>Insufficient Encryption<br>Insufficient Encryption Key Size,<br>Read Not Permitted,<br>Application Error |
| **Read by Group Type Request** | Read by Group Type Response | Yes | Invalid Handle,<br>Request Not Supported,<br>Attribute Not Found,<br>Insufficient Authorization,<br>Insufficient Authentication,<br>Insufficient Encryption<br>Insufficient Encryption Key Size<br>Read Not Permitted,<br>Unsupported Group Type,<br>Application Error |

*Table 3.38: Attribute Request and Response Summary*

| Attribute PDU Method | Successful Response | Error Response Allowed | Error Response Error Codes |
|---|---|---|---|
| **Write Request** | Write Response | Yes | Invalid Handle<br>Request Not Supported,<br>Insufficient Authorization,<br>Insufficient Authentication,<br>Insufficient Encryption<br>Insufficient Encryption Key Size,<br>Write Not Permitted,<br>Invalid Attribute Value Length,<br>Application Error |
| **Write Command** | N/A | No | |
| **Signed Write Command** | N/A | No | |
| **Prepare Write Request** | Prepare Write Response | Yes | Invalid Handle<br>Request Not Supported,<br>Insufficient Authorization,<br>Insufficient Authentication,<br>Write Not Permitted,<br>Prepare Queue Full,<br>Insufficient Encryption<br>Insufficient Encryption Key Size.<br>Application Error |
| **Execute Write Request** | Execute Write Response | Yes | Application Error<br>Invalid Offset<br>Invalid Attribute Value Length |
| **Handle Value Notification** | N/A | No | |
| **Handle Value Indication** | Handle Value Con-firmation | No | |

*Table 3.38: Attribute Request and Response Summary*

# 4 SECURITY CONSIDERATIONS

The attribute protocol can be used to access information that may require both authorization and an authenticated and encrypted physical link before an attribute can be read or written.

If such a request is issued when the client has not been authorized to access this information, the server shall send an *Error Response* with the error code set to «Insufficient Authorization». The authorization requirements for access to a given attribute are not defined in this specification. Each device implementation will determine how authorization occurs. Authorization procedures are defined in GAP, and may be further refined in a higher layer specification.

If such a request is issued when the physical link is unauthenticated, the server shall send an *Error Response* with the error code set to «Insufficient Authentication». A client wanting to read or write this attribute can then request that the physical link be authenticated, and once this has been completed, send the request again.

The attribute protocol can be used to notify or indicate the value of an attribute that may require an authenticated and encrypted physical link before an attribute notification or indication is performed. A server wanting to notify or indicate this attribute can then request that the physical link be authenticated, and once this has been completed, send the notification or indication.

The list of attributes that a device supports is not considered private or confidential information, and therefore the *Find Information Request* shall always be permitted. This implies that an «Insufficient Authorization» or «Insufficient Authentication» error code shall not be used in an *Error Response* for a *Find Information Request*.

For example, an attribute value may be allowed to be read by any device, but only written by an authenticated device. An implementation should take this into account, and not assume that just because it can read an attribute’s value, it will also be able to write the value. Similarly, just because an attribute value can be written, does not mean that an attribute value can also be read. Each individual attribute could have different security requirements.

When a client accesses an attribute, the order of checks that are performed on the server will have security implications. A server shall check authentication and authorization requirements before any other check is performed.

Note: For example, if the authentication and authorization requirement checks are not performed first then the size of an attribute could be determined by performing repeated read blob requests on an attribute that a client does not have access to, because either an «Invalid Offset» error code or «Insufficient Authentication» error codes would be returned.

# 5 ACRONYMS AND ABBREVIATIONS

| Abbreviation or Acronym | Meaning |
|---|---|
| MTU | Maximum Transmission Unit |
| PDU | Protocol Data Unit |
| SDP | Service Discovery Protocol |
| UUID | Universally Unique IDentifier |

*Table 5.1: Abbreviations and Acronyms*

# GENERIC ATTRIBUTE PROFILE (GATT)

***This specification defines the Generic Attribute Profile that describes a service framework using the Attribute Protocol for discovering services, and for reading and writing characteristic values on a peer device.***


Ford Ex. 1052

# CONTENTS

**1 Introduction ..... 524**
1.1 Scope ..... 524
1.2 Profile Dependency ..... 524
1.3 Conformance ..... 524
1.4 Bluetooth Specification Release Compatibility ..... 525
1.5 Conventions ..... 525
**2 Profile Overview ..... 526**
2.1 Protocol Stack ..... 526
2.2 Configurations and Roles ..... 526
2.3 User Requirements and Scenarios ..... 527
2.4 Profile Fundamentals ..... 528
2.5 Attribute Protocol ..... 528
2.5.1 Overview ..... 528
2.5.2 Attribute Caching ..... 529
2.5.3 Attribute Grouping ..... 531
2.5.4 UUIDs ..... 531
2.6 GATT Profile Hierarchy ..... 531
2.6.1 Overview ..... 531
2.6.2 Service ..... 532
2.6.3 Included Services ..... 533
2.6.4 Characteristic ..... 533
2.7 Configured Broadcast ..... 533
**3 Service Interoperability Requirements ..... 535**
3.1 Service Definition ..... 535
3.2 Include Definition ..... 536
3.3 Characteristic Definition ..... 536
3.3.1 Characteristic Declaration ..... 537
3.3.1.1 Characteristic Properties ..... 538
3.3.1.2 Characteristic Value Attribute Handle ..... 538
3.3.1.3 Characteristic UUID ..... 538
3.3.2 Characteristic Value Declaration ..... 539
3.3.3 Characteristic Descriptor Declarations ..... 539
3.3.3.1 Characteristic Extended Properties ..... 540
3.3.3.2 Characteristic User Description ..... 540
3.3.3.3 Client Characteristic Configuration ..... 541
3.3.3.4 Server Characteristic Configuration ..... 542
3.3.3.5 Characteristic Presentation Format ..... 543
3.3.3.6 Characteristic Aggregate Format ..... 546
3.4 Summary of GATT Profile Attribute Types ..... 547

**4 GATT Feature Requirements ..... 548**
4.1 Overview ..... 548
4.2 Feature Support and Procedure Mapping ..... 548
4.3 Server Configuration ..... 550
4.3.1 Exchange MTU ..... 550
4.4 Primary Service Discovery ..... 551
4.4.1 Discover All Primary Services ..... 551
4.4.2 Discover Primary Service by Service UUID ..... 552
4.5 Relationship Discovery ..... 554
4.5.1 Find Included Services ..... 554
4.6 Characteristic Discovery ..... 556
4.6.1 Discover All Characteristics of a Service ..... 556
4.6.2 Discover Characteristics by UUID ..... 557
4.7 Characteristic Descriptor Discovery ..... 558
4.7.1 Discover All Characteristic Descriptors ..... 558
4.8 Characteristic Value Read ..... 560
4.8.1 Read Characteristic Value ..... 560
4.8.2 Read Using Characteristic UUID ..... 560
4.8.3 Read Long Characteristic Values ..... 561
4.8.4 Read Multiple Characteristic Values ..... 562
4.9 Characteristic Value Write ..... 563
4.9.1 Write Without Response ..... 563
4.9.2 Signed Write Without Response ..... 564
4.9.3 Write Characteristic Value ..... 565
4.9.4 Write Long Characteristic Values ..... 566
4.9.5 Reliable Writes ..... 567
4.10 Characteristic Value Notification ..... 569
4.10.1 Notifications ..... 570
4.11 Characteristic Value Indications ..... 570
4.11.1 Indications ..... 570
4.12 Characteristic Descriptors ..... 571
4.12.1 Read Characteristic Descriptors ..... 571
4.12.2 Read Long Characteristic Descriptors ..... 572
4.12.3 Write Characteristic Descriptors ..... 573
4.12.4 Write Long Characteristic Descriptors ..... 574
4.13 GATT Procedure Mapping to ATT Protocol Opcodes ..... 575
4.14 Procedure Timeouts ..... 578
**5 L2CAP Interoperability Requirements ..... 579**
5.1 BR/EDR L2CAP Interoperability Requirements ..... 579

5.1.1 ATT_MTU ..... 579
5.1.2 BR/EDR Channel Requirements ..... 579
5.1.3 BR/EDR Channel Establishment Collisions ..... 579
5.2 LE L2CAP Interoperability Requirements ..... 580
5.2.1 ATT_MTU ..... 580
5.2.2 LE Channel Requirements ..... 580
**6 GAP Interoperability Requirements ..... 582**
6.1 BR/EDR GAP Interoperability Requirements ..... 582
6.1.1 Connection Establishment ..... 582
6.2 LE GAP Interoperability Requirements ..... 582
6.2.1 Connection Establishment ..... 582
6.2.2 Profile Roles ..... 582
6.3 Disconnected Events ..... 582
6.3.1 Notifications and Indications While Disconnected ..... 582
**7 Defined Generic Attribute Profile Service ..... 584**
7.1 Service Changed ..... 584
**8 Security Considerations ..... 586**
8.1 Authentication Requirements ..... 586
8.2 Authorization Requirements ..... 587
**9 SDP Interoperability Requirements ..... 588**
**10 References ..... 589**
**11 Appendix: Example Attribute Server Attributes ..... 590**

# 1 INTRODUCTION

## 1.1 SCOPE

The Generic Attribute Profile (GATT) defines a service framework using the Attribute Protocol. This framework defines procedures and formats of services and their characteristics. The procedures defined include discovering, reading, writing, notifying and indicating characteristics, as well as configuring the broadcast of characteristics.

## 1.2 PROFILE DEPENDENCY

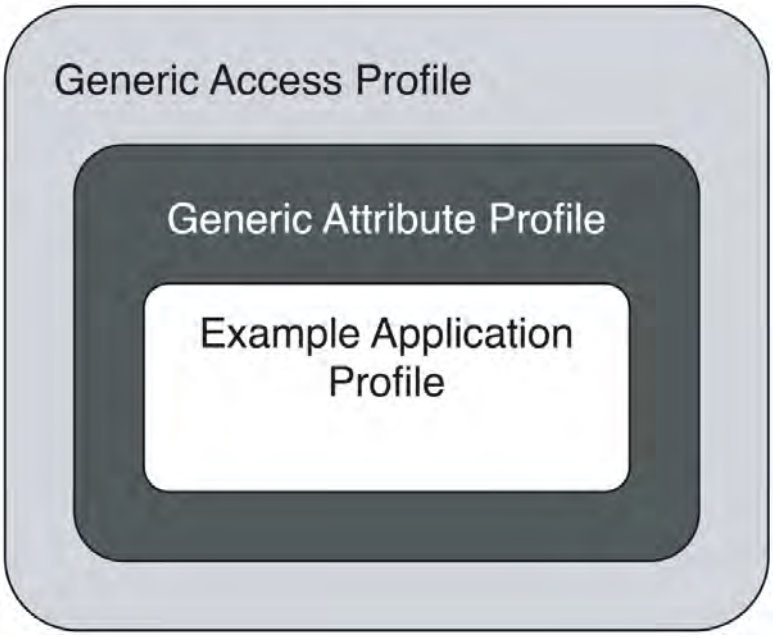



*Figure 1.1: Profile dependencies*

Figure 1.1 depicts the structure and the dependencies of the profiles. A profile is dependent upon another profile if it re-uses parts of that profile by implicitly or explicitly referencing it.

## 1.3 CONFORMANCE

If conformance to this profile is claimed, all capabilities indicated as mandatory for this profile shall be supported in the specified manner (process-mandatory). This also applies for all optional and conditional capabilities for which support is indicated. All mandatory capabilities, and optional and conditional capabilities for which support is indicated, are subject to verification as part of the Bluetooth qualification program.

## 1.4 BLUETOOTH SPECIFICATION RELEASE COMPATIBILITY

This specification can be used with Bluetooth Core Specification Version 1.2 or later when using the profile on the BR/EDR physical link and Bluetooth Core Specification Version 4.0 or later when using the profile on the LE physical link.

## 1.5 CONVENTIONS

In this specification the use of literal terms such as procedure, PDUs, opcodes or function names appear in italics. Specific names of fields in structures, packets, etc. also appear in italics. The use of « » (e.g. «Primary Service») indicate a Bluetooth SIG-defined UUID.

# 2 PROFILE OVERVIEW

The GATT profile is designed to be used by an application or another profile, so that a client can communicate with a server. The server contains a number of attributes, and the GATT Profile defines how to use the Attribute Protocol to discover, read, write and obtain indications of these attributes, as well as configuring broadcast of attributes.

## 2.1 PROTOCOL STACK

Figure 2.1 shows the peer protocols used by this profile.

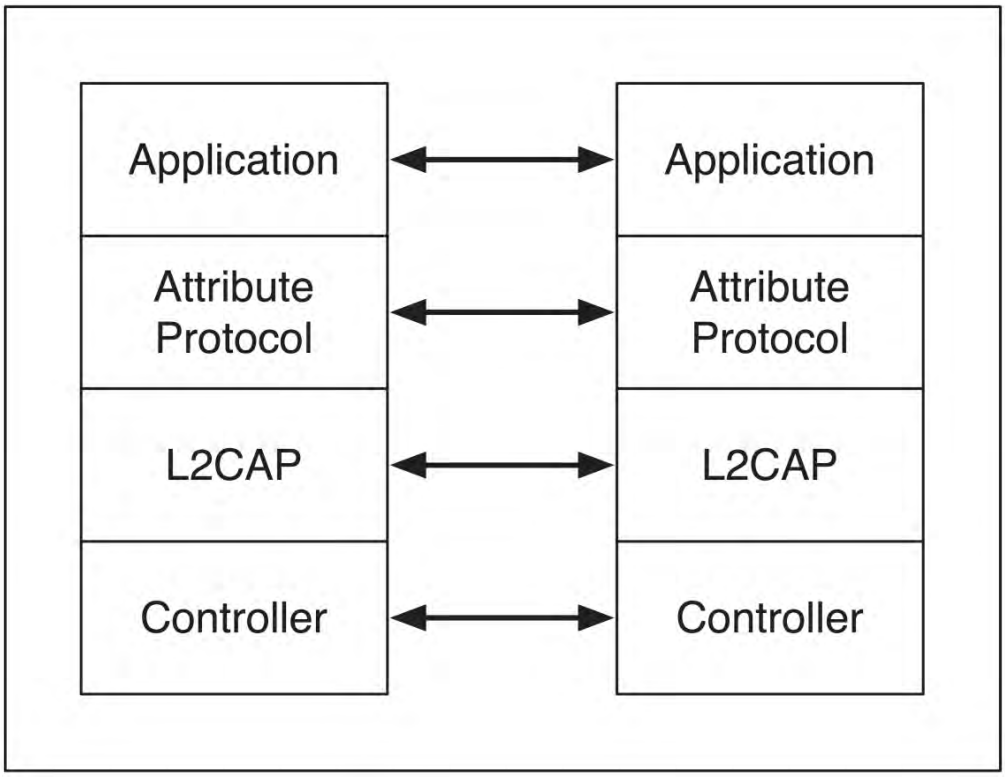



*Figure 2.1: Protocol model*

## 2.2 CONFIGURATIONS AND ROLES

The following roles are defined for devices that implement this profile:

Client—This is the device that initiates commands and requests towards the server and can receive responses, indications and notifications sent by the server.

Server—This is the device that accepts incoming commands and requests from the client and sends responses, indications and notifications to a client.

Note: The roles are not fixed to the device. The roles are determined when a device initiates a defined procedure, and they are released when the procedure ends.

A device can act in both roles at the same time.

An example of configurations illustrating the roles for this profile is depicted in Figure 2.2.

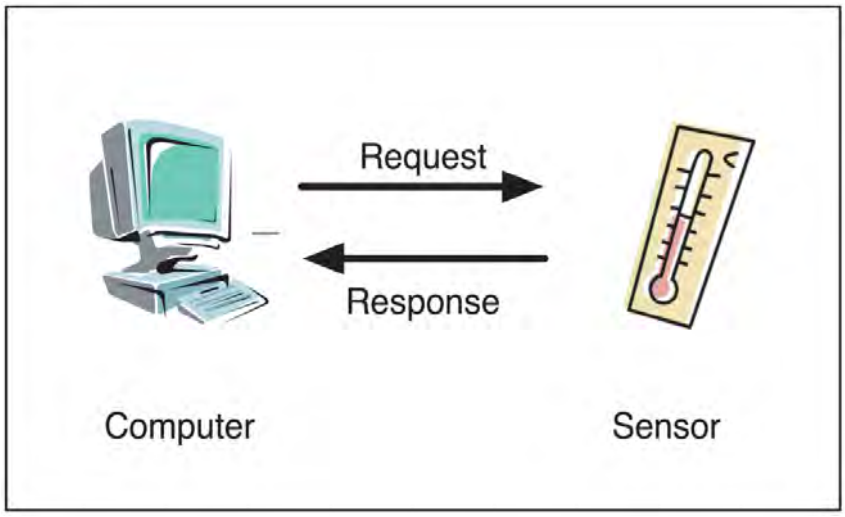



*Figure 2.2: Examples of configuration*

In Figure 2.2, the computer is the temperature service client and the sensor is the temperature service server. The computer initiates procedures to configure the sensor or to read the sensor values. In this example the sensor provides information about the characteristics the sensor device exposes as part of the temperature service and may permit some characteristics to be written. Also, the sensor responds to read requests with the appropriate values.

## 2.3 USER REQUIREMENTS AND SCENARIOS

The following scenarios are covered by this profile:

- Exchanging configuration
- Discovery of services and characteristics on a device
- Reading a characteristic value
- Writing a characteristic value
- Notification of a characteristic value
- Indication of a characteristic value

## 2.4 PROFILE FUNDAMENTALS

This profile can be used over any physical link, using the Attribute Protocol L2CAP channel, known as the ATT Bearer. Here is a brief summary of lower layer requirements communication between the client and the server.

- An ATT Bearer is established using “Channel Establishment” as defined in Section 6.
- The profile roles are not tied to the controller master/slave roles.
- On an LE Physical link, use of security features such as authorization, authentication and encryption are optional. On a BR/EDR physical link encryption is mandatory.
- Multi-octet fields within the GATT Profile shall be sent least significant octet first (little endian).

## 2.5 ATTRIBUTE PROTOCOL

The GATT Profile requires the implementation of the Attribute Protocol 11 and the required Attribute opcodes indicated in Section 4.2.

### 2.5.1 Overview

The GATT Profile uses the Attribute Protocol to transport data in the form of commands, requests, responses, indications, notifications and confirmations between devices. This data is contained in Attribute Protocol PDUs.

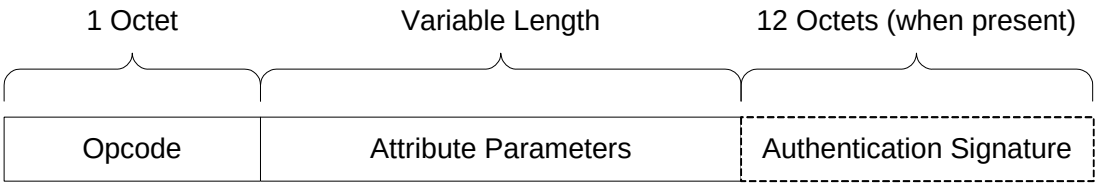



*Figure 2.3: Attribute Protocol PDU*

The *Opcode* contains the specific command, request, response, indication, notification or confirmation opcode and a flag for authentication. The *Attribute Parameters* contains data for the specific command or request or the data returned in a response, indication or notification. The *Authentication Signature* is optional and is described in [Vol. 6] Part F, Section 3.3.1.

Attribute Protocol commands and requests act on values stored in Attributes on the server device. An Attribute is composed of four parts: *Attribute Handle*, *Attribute Type*, *Attribute Value,* and *Attribute Permissions*. Figure 2.4 shows a logical representation of an Attribute. The actual representation for a given implementation is specific to that implementation.

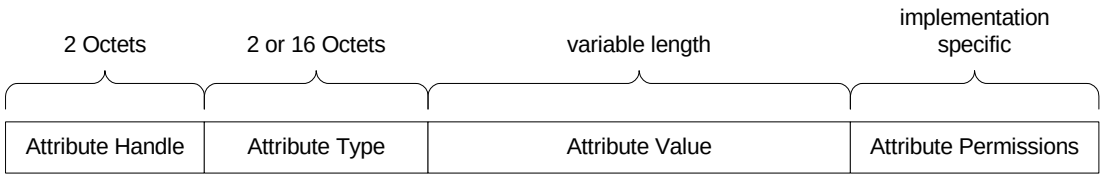



*Figure 2.4: Logical Attribute Representation*

The *Attribute Handle* is an index corresponding to a specific Attribute. The *Attribute Type* is a UUID that describes the *Attribute Value*. The *Attribute Value* is the data described by the *Attribute Type* and indexed by the *Attribute Handle*. The Attributes are ordered by increasing *Attribute Handle* values. *Attribute Handle* values may begin at any value between 0x0001 and 0xFFFF. Although the *Attribute Handle* values are in increasing order, following *Attribute Handle* values may differ by more than one. That is to say there may be gaps between successive *Attribute Handles*.

*Attribute Permissions* is part of the Attribute that cannot be read from or written to using the Attribute Protocol. It is used by the server to determine whether read or write access is permitted for a given attribute. *Attribute Permissions* are established by the GATT profile, a higher layer profile or are implementation specific if not specified.

### 2.5.2 Attribute Caching

Attribute caching is an optimization that allows the client to discover the Attribute information such as *Attribute Handles* used by the server once and use the same Attribute information across reconnections without rediscovery. Without caching the Attribute information, the client shall rediscover the *Attribute* information at each reconnection. With caching, time is saved and a significant amount of packets exchanged between the client and server is not required. The Attribute information that shall be cached by a client is the *Attribute Handle*s of all server attributes and the GATT service characteristics values.

*Attribute Handles* used by the server should not change over time. This means that once an *Attribute Handle* is discovered by a client the *Attribute Handle* for that Attribute should not be changed.

Some circumstances may cause servers to change the *Attribute Handles* used for services, perhaps due to a factory reset or a firmware upgrade procedure being performed. The following is only required on the server if the services on the server can be added, modified or removed. If GATT based services on the server cannot be changed during the usable lifetime of the device, the *Services Changed* characteristic shall not exist on the server and the client does not need to ever perform service discovery after the initial service discovery for that server.

To support caching when a server supports changes in GATT based services, an indication is sent by the server to clients when a service is added, removed, or modified on the server. A GATT based service is considered modified if the binding of the *Attribute Handles* to the associated Attributes grouped within a service definition are changed. Any change to the GATT service definition characteristic values other than the *Service Change* characteristic value itself shall also be considered a modification.

For clients that have a trusted relationship (i.e. bond) with the server, the attribute cache is valid across connections. For clients with a trusted relationship and not in a connection when a service change occurs, the server shall send an indication when the client reconnects to the server. For clients that do not have a trusted relationship with the server, the attribute cache is valid only during the connection. Clients without a trusted relationship shall receive an indication when the service change occurs only during the current connection.

Note: Clients without a trusted relationship must perform service discovery on each connection if the server supports the *Services Changed* characteristic.

The server shall send a *Handle Value Indication* containing the range of affected *Attribute Handles* that shall be considered invalid in the client's attribute cache. The start *Attribute Handle* shall be the start *Attribute Handle* of the service definition containing the change and the end *Attribute Handle* shall be the last *Attribute Handle* of the service definition containing the change. The value in the indication is composed of two 16-bit *Attribute Handle*s concatenated to indicate the affected *Attribute Handle* range.

Note: A server may set the affected *Attribute Handle* range to 0x0001 to 0xFFFF to indicate to the client to rediscover the entire set of *Attribute Handles* on the server.

The client, upon receiving a Handle Value Indication containing the range of affected Attribute Handles, shall consider the attribute cache invalid over the affected Attribute Handle range. Any outstanding request transaction shall be considered invalid if the Attribute Handle is contained within the affected Attribute Handle range. The client must perform service discovery before the client uses any service that has an attribute within the affected Attribute Handle range.

Once the server has received the Handle Value Confirmation, the server can consider the client to be aware of the updated Attribute Handles.

The client shall consider the affected *Attribute Handle* range to be invalid in its attribute cache and perform the discovery procedures to restore the attribute cache. The server shall store service changed information for all bonded devices.

### 2.5.3 Attribute Grouping

Generic attribute profile defines grouping of attributes for three attribute types: «Primary Service», «Secondary Service» and «Characteristic». A group begins with a declaration, and ends as defined in Section 3.1 for services and Section 3.3 for characteristics.Not all of the grouping attributes can be used in the ATT Read By Group Type Request. The «Primary Service» and «Secondary Service» grouping types may be used in the Read By Group Type Request. The «Characteristic» grouping type shall not be used in the ATT Read By Group Type Request.

### 2.5.4 UUIDs

All 16-bit UUIDs shall be contained in exactly 2 octets. All 128-bit UUIDs shall be contained in exactly 16 octets.

## 2.6 GATT PROFILE HIERARCHY

### 2.6.1 Overview

The GATT Profile specifies the structure in which profile data is exchanged. This structure defines basic elements such as services and characteristics, used in a profile. All of the elements are contained by Attributes. Attributes used in the Attribute Protocol are containers that carry this profile data.

The top level of the hierarchy is a profile. A profile is composed of one or more services necessary to fulfill a use case. A service is composed of characteristics or references to other services. Each characteristic contains a value and may contain optional information about the value. The service and characteristic and the components of the characteristic (i.e. value and descriptors) contain the profile data and are all stored in Attributes on the server.

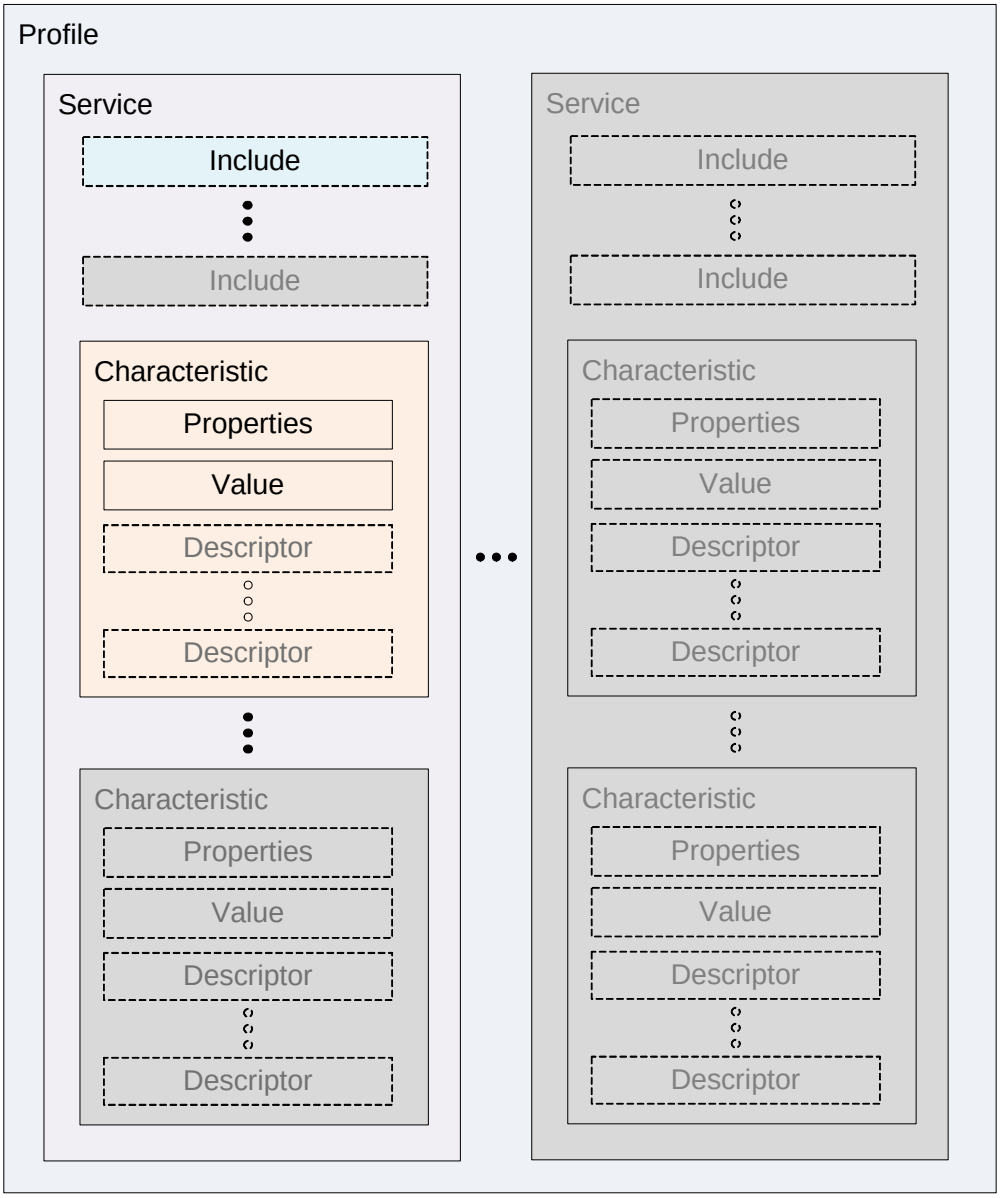



*Figure 2.5: GATT Profile hierarchy*

### 2.6.2 Service

A service is a collection of data and associated behaviors to accomplish a particular function or feature. In GATT, a service is defined by its service definition. A service definition may contain referenced services, mandatory characteristics and optional characteristics.

To maintain backward compatibility with earlier clients, later versions of a service definition can only add new referenced services or optional characteristics. Later versions of a service definition are forbidden from changing behaviors from previous versions of the service definition.

There are two types of services: primary service and secondary service. A primary service is a service that exposes the primary usable functionality of this

device. A primary service can be included by another service. Primary services can be discovered using Primary Service Discovery procedures. A secondary service is a service that is only intended to be referenced from a primary service or another secondary service or other higher layer specification. A secondary service is only relevant in the context of the entity that references it.

The determination of whether a service is either a primary or secondary service can be mandated by a higher layer specification.

Services may be used in one or more higher layer specifications to fulfill a particular use case.

The service definition is described in Section 3.1.

### 2.6.3 Included Services

An included service is a method to reference another service definition existing on the server into the service being defined. To include another service, an include definition is used at the beginning of the service definition. When a service definition uses an include definition to reference the included service, the entire included service definition becomes part of the new service definition. This includes all the included services and characteristics of the included service. The included service still exists as an independent service. A service that is included by another service shall not be changed by the act of inclusion or by the including service. There are no limits to the number of include definitions or the depth of nested includes in a service definition.

The include definition is described in Section 3.2.

### 2.6.4 Characteristic

A characteristic is a value used in a service along with properties and configuration information about how the value is accessed and information about how the value is displayed or represented. In GATT, a characteristic is defined by its characteristic definition. A characteristic definition contains a characteristic declaration, characteristic properties, and a value and may contain descriptors that describe the value or permit configuration of the server with respect to the characteristic.

The characteristic definition is described in Section 3.3.

## 2.7 CONFIGURED BROADCAST

For LE physical links, Configured Broadcast is a method for a client to indicate to a server which *Characteristic Value* shall be broadcast in the advertising data when the server is executing the broadcast mode procedure.

To configure a *Characteristic Value* to be broadcast by the server when in broadcast mode, the client sets the broadcast configuration bit described in

Section 3.3.3.3. The frequency of the broadcast is part of the service or characteristic behavior definition.

# 3 SERVICE INTEROPERABILITY REQUIREMENTS

## 3.1 SERVICE DEFINITION

A service definition shall contain a service declaration and may contain include definitions and characteristic definitions. The service definition ends before the next service declaration or after the maximum *Attribute Handle* is reached. Service definitions appear on the server in an order based on *Attribute Handle*.

All include definitions and characteristic definitions contained within the service definition are considered to be part of the service. All include definitions shall immediately follow the service declaration and precede any characteristic definitions. A service definition may have zero or more include definitions. All characteristic definitions shall be immediately following the last include definition or in the event of no include definitions, immediately following the service declaration. A service definition may have zero or more characteristic definitions. There is no upper limit for include or characteristic definitions.

A service declaration is an Attribute with the *Attribute Type* set to the UUID for «Primary Service» or «Secondary Service». The *Attribute Value* shall be the 16-bit Bluetooth UUID or 128-bit UUID for the service, known as the service UUID. A client shall support the use of both 16-bit and 128-bit UUIDs. A client may ignore any service definition with an unknown service UUID. An unknown service UUID is a UUID for an unsupported service. The *Attribute Permissions* shall be read-only and shall not require authentication or authorization.

When multiple services exist, services definitions with service declarations using 16-bit Bluetooth UUID should be grouped together (i.e. listed sequentially) and services definitions with service declarations using 128-bit UUID should be grouped together.

| Attribute Handle | Attribute Type | Attribute Value | Attribute Permission |
|---|---|---|---|
| 0xNNNN | 0x2800 – UUID for «Primary Service» OR 0x2801 for «Secondary Service»] | 16-bit Bluetooth UUID or 128-bit UUID for Service | Read Only, No Authentication, No Authorization |

*Table 3.1: Service Declaration*

A device or higher level specification may have multiple service definitions and may have multiple service definitions with the same service UUID.

All Attributes on a Server shall either contain a service declaration or exist within a service definition.

Service definitions contained in a server may appear in any order; a client shall not assume the order of service definitions on a server.

## 3.2 INCLUDE DEFINITION

An include definition shall contain only one include declaration.

The include declaration is an Attribute with the *Attribute Type* set to the UUID for «Include». The *Attribute Value* shall be set to the included service *Attribute Handle,* the End Group Handle, and the *service UUID*. The Service UUID shall only be present when the UUID is a 16-bit Bluetooth UUID.The *Attribute Permissions* shall be read only and not require authentication or authorization.

| Attribute Handle | Attribute Type | Attribute Value | | | Attribute Permission |
|---|---|---|---|---|---|
| 0xNNNN | 0x2802 – UUID for «Include» | Included Service Attribute Handle | End Group Handle | Service UUID | Read Only, No Authentication, No Authorization |

*Table 3.2: Include Declaration*

A server shall not contain a service definition with an include definition to another service that references the original service. This applies to each of the services the included definition references. This is referred to as a circular reference.

If the client detects a circular reference or detects nested include declarations to a greater level than it expects, it should terminate the ATT Bearer.

## 3.3 CHARACTERISTIC DEFINITION

A characteristic definition shall contain a characteristic declaration, a Characteristic Value declaration and may contain characteristic descriptor declarations. A characteristic definition ends at the start or the next characteristic declaration or service declaration or after the maximum *Attribute Handle*. Characteristic definitions appear on the server within a service definition in an order based on *Attribute Handle*.

Each declaration above is contained in a separate Attribute. The two required declarations are the characteristic declaration and the Characteristic Value declaration. The Characteristic Value declaration shall exist immediately following the characteristic declaration. Any optional characteristic descriptor declarations are placed after the Characteristic Value declaration. The order of the optional characteristic descriptor declarations is not significant.

A characteristic definition may be defined to concatenate several Characteristic Values into a single aggregated Characteristic Value. This may be used to optimize read and writes of multiple Characteristic Values through the reading and writing of a single aggregated Characteristic Value. This type of characteristic definition is the same as a normal characteristic definition. The characteristic declaration shall use a characteristic UUID that is unique to the aggregated

characteristic definition. The aggregated characteristic definition may also contain a characteristic aggregate format descriptor that describes the display format of the aggregated Characteristic Value.

### 3.3.1 Characteristic Declaration

A characteristic declaration is an Attribute with the Attribute Type set to the UUID for «Characteristic» and *Attribute Value* set to the Characteristic Properties, Characteristic Value *Attribute Handle* and Characteristic UUID. The Attribute Permissions shall be readable and not require authentication or authorization.

The characteristic declaration *Attribute Value* shall not change while the server has a trusted relationship with any client.

| **Attribute Handle** | **Attribute Type s** | **Attribute Value** | | | **Attribute Permissions** |
|---|---|---|---|---|---|
| 0xNNNN | 0x2803–UUID for «Characteristic» | Characteristic Properties | Characteristic Value Attribute Handle | Characteristic UUID | Read Only, No Authentication, No Authorization |

*Table 3.3: Characteristic declaration*

The *Attribute Value* of a characteristic declaration is read only.

| **Attribute Value** | **Size** | **Description** |
|---|---|---|
| **Characteristic Properties** | 1 octets | Bit field of characteristic properties |
| **Characteristic Value Handle** | 2 octets | Handle of the Attribute containing the value of this characteristic |
| **Characteristic UUID** | 2 or 16 octets | 16-bit Bluetooth UUID or 128-bit UUID for Characteristic Value |

*Table 3.4: Attribute Value Field in characteristic declaration*

A service may have multiple characteristic definitions with the same Characteristic UUID.

Within a service definition, some characteristics may be mandatory and those characteristics shall be located after the include declarations and before any optional characteristics within the service definition. A client shall not assume any order of those characteristics that are mandatory or any order of those characteristics that are optional within a service definition. Whenever possible and within the requirements stated earlier, characteristics definitions with characteristic declarations using 16-bit Bluetooth UUIDs should be group together (i.e. listed sequentially) and characteristics definitions with characteristic declarations using 128-bit UUIDs should be grouped together.

### 3.3.1.1 Characteristic Properties

The Characteristic Properties bit field determines how the Characteristic Value can be used, or how the characteristic descriptors (see Section 3.3.3) can be accessed. If the bits defined in Table 3 5 are set, the action described is permitted. Multiple Characteristic Properties can be set.

These bits shall be set according to the procedures this characteristic supports, without regard to security requirements.

| Properties | Value | Description |
|---|---|---|
| Broadcast | 0x01 | If set, permits broadcasts of the Characteristic Value using Characteristic Configuration Descriptor |
| Read | 0x02 | If set, permits reads of the Characteristic Value using procedures defined in Section 4.8 |
| Write Without Response | 0x04 | If set, permit writes of the Characteristic Value without response using procedures defined in Section 4.9.1. |
| Write | 0x08 | If set, permits writes of the Characteristic Value with response using procedures defined in Section 4.9.3 or Section 4.9.4. |
| Notify | 0x10 | If set, permits notifications of a Characteristic Value without acknowledgement using the procedure defined in Section 4.10. |
| Indicate | 0x20 | If set, permits indications of a Characteristic Value with acknowledgement using the procedure defined in Section 4.11. |
| Authenticated Signed Writes | 0x40 | If set, permits signed writes to the Characteristic Value using the procedure defined in Section 4.9.2. |
| Extended Properties | 0x80 | If set, additional characteristic properties are defined in the Characteristic Extended Properties Descriptor defined in Section 3.3.3.1. |

*Table 3.5: Characteristic Properties bit field*

### 3.3.1.2 Characteristic Value Attribute Handle

The Characteristic Value *Attribute Handle* field is the *Attribute Handle* of the Attribute that contains the *Characteristic Value*.

### 3.3.1.3 Characteristic UUID

The *Characteristic UUID* field is a 16-bit Bluetooth UUID or 128-bit UUID that describes the type of *Characteristic Value*. A client shall support the use of both

16-bit and 128-bit *Characteristic UUIDs*. A client may ignore any characteristic definition with an unknown *Characteristic UUID*. An unknown characteristic UUID is a UUID for an unsupported characteristic.

### 3.3.2 Characteristic Value Declaration

The *Characteristic Value* declaration contains the value of the characteristic. It is the first Attribute after the characteristic declaration. All characteristic definitions shall have a *Characteristic Value* declaration.

A Characteristic Value declaration is an Attribute with the Attribute Type set to the 16-bit Bluetooth or 128-bit UUID for the Characteristic Value used in the characteristic declaration. The *Attribute Value* is set to the *Characteristic Value*. The *Attribute Permissions* are specified by the service or may be implementation specific if not specified otherwise.

| Attribute Handle | Attribute Type | Attribute Value | Attribute Permissions |
|---|---|---|---|
| 0xNNNN | 0xuuuu – 16-bit Bluetooth UUID or 128-bit UUID for Characteristic UUID | Characteristic Value | Higher layer profile or implementation specific |

*Table 3.6: Characteristic Value declaration*

### 3.3.3 Characteristic Descriptor Declarations

Characteristic descriptors are used to contain related information about the *Characteristic Value*. The GATT profile defines a standard set of characteristic descriptors that can be used by higher layer profiles. Higher layer profiles may define additional characteristic descriptors that are profile specific. Each characteristic descriptor is identified by the characteristic descriptor UUID. A client shall support the use of both 16-bit and 128-bit characteristic descriptor UUIDs. A client may ignore any characteristic descriptor declaration with an unknown characteristic descriptor UUID. An unknown characteristic descriptor UUID is a UUID for an unsupported characteristic descriptor.

Characteristic descriptors if present within a characteristic definition shall follow the *Characteristic Value* declaration. The characteristic descriptor declaration may appear in any order within the characteristic definition. The client shall not assume the order in which a characteristic descriptor declaration appears in a characteristic definition following the *Characteristic Value* declaration.

Characteristic descriptor declaration permissions are defined by a higher layer profile or are implementation specific. A client shall not assume all characteristic descriptor declarations are readable.

### 3.3.3.1 Characteristic Extended Properties

The *Characteristic Extended Properties* declaration is a descriptor that defines additional *Characteristic Properties*. If the *Extended Properties* bit of the *Characteristic Properties* is set then this characteristic descriptor shall exist. The characteristic descriptor may occur in any position within the characteristic definition after the Characteristic Value. Only one *Characteristic Extended Properties* declaration shall exist in a characteristic definition.

The characteristic descriptor is contained in an Attribute and the *Attribute Type* shall be set to the UUID for «Characteristic Extended Properties» and the *Attribute Value* shall be the *Characteristic Extended Properties Bit Field*. The *Attribute Permissions* shall be readable without authentication and authorization being required.

| Attribute Handle | Attribute Type | Attribute Value | Attribute Permissions |
|---|---|---|---|
| 0xNNNN | 0x2900 – UUID for «Characteristic Extended Properties» | Characteristic Extended Properties Bit Field | Read Only,<br>No Authentication,<br>No Authorization |

*Table 3.7: Characteristic Extended Properties declaration*

The *Characteristic Extended Properties* bit field describes additional properties on how the Characteristic Value can be used, or how the characteristic descriptors (see Section 3.3.3.3) can be accessed. If the bits defined in Table 3 8 are set, the action described is permitted. *Multiple Characteristic Properties* can be set.

| Properties | Value | Description |
|---|---|---|
| Reliable Write | 0x0001 | If set, permits reliable writes of the Characteristic Value using the procedure defined in Section 4.9.5. |
| Writable Auxiliaries | 0x0002 | If set, permits writes to the characteristic descriptor defined in Section 3.3.3.2 |
| Reserved for Future Use | 0xFFFC | Reserved for Future Use |

*Table 3.8: Characteristic Extended Properties bit field*

### 3.3.3.2 Characteristic User Description

The *Characteristic User* Description declaration is an optional characteristic descriptor that defines a UTF-8 string of variable size that is a user textual description of the *Characteristic Value*. If the *Writable Auxiliary* bit of the *Characteristic Properties* is set then this characteristic descriptor can be written. The characteristic descriptor may occur in any position within the characteristic

definition after the *Characteristic Value*. Only one *Characteristic User Description* declaration shall exist in a characteristic definition.

The characteristic descriptor is contained in an Attribute and the *Attribute Type* shall be set to the UUID for «Characteristic User Description» and the *Attribute Value* shall be set to the characteristic user description UTF-8 string. The *Attribute Permissions* are specified by the profile or may be implementation specific if not specified otherwise.

| Attribute Handle | Attribute Type | Attribute Value | Attribute Permissions |
|---|---|---|---|
| 0xNNNN | 0x2901 – UUID for «Characteristic User Description» | Characteristic User Description UTF-8 String | Higher layer profile or implementation specific |

*Table 3.9: Characteristic User Description declaration*

### 3.3.3.3 Client Characteristic Configuration

The *Client Characteristic Configuration* declaration is an optional characteristic descriptor that defines how the characteristic may be configured by a specific client. The Client Characteristic Configuration descriptor value shall be persistent across connections for bonded devices. The Client Characteristic Configuration descriptor value shall be set to the default value at each connection with non-bonded devices.The characteristic descriptor value is a bit field. When a bit is set, that action shall be enabled, otherwise it will not be used. The *Client Characteristic Configuration* descriptor may occur in any position within the characteristic definition after the Characteristic Value. Only one *Client Characteristic Configuration* declaration shall exist in a characteristic definition.

A client may write this configuration descriptor to control the configuration of this characteristic on the server for the client. Each client has its own instantiation of the *Client Characteristic Configuration*. Reads of the *Client Characteristic Configuration* only shows the configuration for that client and writes only affect the configuration of that client. Authentication and authorization may be required by the server to write the configuration descriptor. The *Client Characteristic Configuration* declaration shall be readable and writable.

The characteristic descriptor is contained in an Attribute. The *Attribute Type* shall be set to the UUID for «Client Characteristic Configuration». The *Attribute Value* shall be set to the characteristic descriptor value. The *Attribute Permissions* are specified by the profile or may be implementation specific if not specified otherwise.

The following Client Characteristic Configuration bits are defined:

The default value for the *Client Characteristic Configuration* descriptor value shall be 0x0000.

| Attribute Handle | Attribute Type | Attribute Value | Attribute Permissions |
|---|---|---|---|
| 0xNNNN | 02902 – UUID for «Client Characteristic Configuration» | Characteristic Configuration Bits | Readable with no authentication or authorization.<br>Writable with authentication and authorization defined by a higher layer specification or is implementation specific. |

*Table 3.10: Client Characteristic Configuration declaration*

| Configuration | Value | Description |
|---|---|---|
| Notification | 0x0001 | The Characteristic Value shall be notified. |
| Indication | 0x0002 | The Characteristic Value shall be indicated. |
| Reserved for Future Use | 0xFFF4 | Reserved for future use. |

*Table 3.11: Client Characteristic Configuration bit field definition*

### *3.3.3.4 Server Characteristic Configuration*

The *Server Characteristic Configuration* declaration is an optional characteristic descriptor that defines how the characteristic may be configured for the server. The characteristic descriptor value is a bit field. When a bit is set, that action shall be enabled, otherwise it will not be used. The *Server Characteristic Configuration* descriptor may occur in any position within the characteristic definition after the *Characteristic Value*. Only one *Server Characteristic Configuration* declaration shall exist in a characteristic definition. The *Server Characteristic Configuration* declaration shall be readable and writable.

A client may write this configuration descriptor to control the configuration of this characteristic on the server for all clients. There is a single instantiation of the *Server Characteristic Configuration* for all clients. Reads of the *Server Characteristic Configuration* shows the configuration all clients and writes affect the configuration for all clients. Authentication and authorization may be required by the server to write the configuration descriptor.

The characteristic descriptor is contained in an Attribute. The *Attribute Type* shall be set to the UUID for «Server Characteristic Configuration». The *Attribute Value* shall be set to the characteristic descriptor value. The *Attribute Permissions* are specified by the profile or may be implementation specific if not specified otherwise.

The following *Server Characteristic Configuration* bits are defined:

| Attribute Handle | Attribute Type | Attribute Value | Attribute Permissions |
|---|---|---|---|
| 0xNNNN | 0x2903 – UUID for «Server Characteristic Configuration» | Characteristic Configuration Bits | Readable with no authentication or authorization.<br>Writable with authentication and authorization defined by a higher layer specification or is implementation specific. |

*Table 3.12: Server Characteristic Configuration declaration*

| Configuration | Value | Description |
|---|---|---|
| Broadcast | 0x0001 | The Characteristic Value shall be broadcast when the server is in the broadcast procedure if advertising data resources are available. |
| Reserved for Future Use | 0xFFF2 | Reserved for future use. |

*Table 3.13: Server Characteristic Configuration bit field definition*

### *3.3.3.5 Characteristic Presentation Format*

The *Characteristic Presentation Format* declaration is an optional characteristic descriptor that defines the format of the *Characteristic Value*. The characteristic descriptor may occur in any position within the characteristic definition after the *Characteristic Value*. If more than one *Characteristic Presentation Format* declarations exist, in a characteristic definition, then a *Characteristic Aggregate Format* declaration shall exist as part of the characteristic definition.

The characteristic format value is composed of five parts: format, exponent, unit, name space, and description.

The characteristic descriptor is contained in an Attribute. The *Attribute Type* shall be set to the UUID for «Characteristic Format». The *Attribute Value* shall be set to the characteristic descriptor value. The *Attribute Permissions* shall be read only and not require authentication or authorization.

| Attribute Handle | Attribute Type | Attribute Value | | | | | Attribute Permissions |
|---|---|---|---|---|---|---|---|
| 0xNNNN | 0x2904 – UUID for «Characteristic Format» | Format | Exponent | Unit | Name Space | Description | Read only<br>No Authentication, NO authorization |

*Table 3.14: Characteristic Format declaration*

The definition of the Characteristic Presentation Format descriptor Attribute Value field is the following.

| Field Name | Value Size | Description |
|---|---|---|
| Format | 1 octet | Format of the value of this characteristic. |
| Exponent | 1 octet | Exponent field to determine how the value of this characteristic is further formatted. |
| Unit | 2 octets | The unit of this characteristic as defined in [1] |
| Name Space | 1 octet | The name space of the description as defined in [1] |
| Description | 2 octets | The description of this characteristic as defined in a higher layer profile. |

*Table 3.15: Characteristic Format Value definition*

#### *3.3.3.5.1 Bit Ordering*

The bit ordering used for the Characteristic Format descriptor shall be little-endian.

#### *3.3.3.5.2 Format*

The format field determines how a single value contained in the *Characteristic Value* is formatted. If a format is not a whole number of octets, then the data shall be contained within the least significant bits of the value, and all other bits shall be set to zero on transmission and ignored upon receipt. If the *Characteristic Value* is less than an octet, it occupies an entire octet.

The following format values are defined:

| Format | Short Name | Description | Exponent Value |
|---|---|---|---|
| 0x00 | rfu | Reserved for Future Used | No |
| 0x01 | boolean | unsigned 1-bit; 0 = false, 1 = true | No |
| 0x02 | 2bit | unsigned 2-bit integer | No |
| 0x03 | nibble | unsigned 4-bit integer | No |
| 0x04 | uint8 | unsigned 8-bit integer | Yes |
| 0x05 | uint12 | unsigned 12-bit integer | Yes |
| 0x06 | uint16 | unsigned 16-bit integer | Yes |
| 0x07 | uint24 | unsigned 24-bit integer | Yes |
| 0x08 | uint32 | unsigned 32-bit integer | Yes |
| 0x09 | uint48 | unsigned 48-bit integer | Yes |
| 0x0A | uint64 | unsigned 64-bit integer | Yes |

*Table 3.16: Characteristic Format types*

| Format | Short Name | Description | Exponent Value |
|---|---|---|---|
| 0x0B | uint128 | unsigned 128-bit integer | Yes |
| 0x0C | sint8 | signed 8-bit integer | Yes |
| 0x0D | sint12 | signed 12-bit integer | Yes |
| 0x0E | sint16 | signed 16-bit integer | Yes |
| 0x0F | sint24 | signed 24-bit integer | Yes |
| 0x10 | sint32 | signed 32-bit integer | Yes |
| 0x11 | sint48 | signed 48-bit integer | Yes |
| 0x12 | sint64 | signed 64-bit integer | Yes |
| 0x13 | sint128 | signed 128-bit integer | Yes |
| 0x14 | float32 | IEEE-754 32-bit floating point | No |
| 0x15 | float64 | IEEE-754 64-bit floating point | No |
| 0x16 | SFLOAT | IEEE-11073 16-bit SFLOAT | No |
| 0x17 | FLOAT | IEEE-11073 32-bit FLOAT | No |
| 0x18 | duint16 | IEEE-20601 format | No |
| 0x19 | utf8s | UTF-8 string | No |
| 0x1A | utf16s | UTF-16 string | No |
| 0x1B | struct | Opaque structure | No |
| 0x1C – 0xFF | rfu | Reserved for Future Use | No |

*Table 3.16: Characteristic Format types*

When encoding an IPv4 address, the uint32 Format type shall be used.

When encoding an IPv6 address, the uint128 Format type shall be used.

When encoding a Bluetooth BD_ADDR, the uint48 Format type shall be used.

A duint16 is two uint16 values concatenated together.

### *3.3.3.5.3 Exponent*

The exponent field is used with integer data types to determine how the value is further formatted. The exponent field is only used on integer format types as indicated in the format field in Table 3 16. The exponent field is a signed integer.

$$\text{actual value} = \textit{Characteristic Value} * 10^{\text{Exponent}}$$

As can be seen in the above equation, the actual value is a combination of the Characteristic Value and the value 10 to the power Exponent. This is sometimes known as a fixed point number.

For example, if the Exponent is 2 and the *Characteristic Value* is 23, the actual value would be 2300.

For example, if the Exponent is -3 and the *Characteristic Value* is 3892, the actual value would be 3.892.

#### *3.3.3.5.4 Unit*

The Unit is a UUID as defined in the Assigned Numbers document [1].

#### *3.3.3.5.5 Name Space*

The Name Space field is used to identify the organization as defined in the Assigned Numbers document [1], that is responsible for defining the enumerations for the description field.

#### *3.3.3.5.6 Description*

The Description is an enumerated value as defined in the Assigned Numbers document [1] from the organization identified by the Name Space field.

### *3.3.3.6 Characteristic Aggregate Format*

The *Characteristic Aggregate Format* declaration is an optional characteristic descriptor that defines the format of an aggregated *Characteristic Value*.

The characteristic descriptor may occur in any position within the characteristic definition after the *Characteristic Value*. Only one *Characteristic Aggregate Format* declaration shall exist in a characteristic definition.

The Characteristic Aggregate Format value is composed of a list of Attribute Handles of *Characteristic Presentation Format* declarations, where each *Attribute Handle* points to a *Characteristic Presentation Format* declaration.

The *Attribute Permissions* shall be read only and not require authentication or authorization.

The *List of Attribute Handles* is the concatenation of multiple 16-bit *Attribute Handle* values into a single *Attribute Value*. The list shall contain at least two *Attribute Handle for Characteristic Presentation Format declarations*. The Characteristic Value shall be decomposed by each of the *Characteristic Presentation Format* declarations pointed to by the *Attribute Handles*. The order of the *Attribute Handles* in the list is significant.

| Attribute Handle | Attribute Type | Attribute Value | Attribute Permissions |
|---|---|---|---|
| 0xNNNN | 0x2905 – UUID for «Characteristic Aggregate Format» | List of *Attribute Handles* for the Characteristic Presentation Format Declarations | Read only<br>No authentication<br>No authorization |

*Table 3.17: Characteristic Aggregate Format declaration*

If more than one *Characteristic Presentation Format* declarations exist in a characteristic definition, there shall also be one Characteristic Aggregate Format declaration. The *Characteristic Aggregate Forma*t declaration shall include each *Characteristic Presentation Format* declaration in the characteristic definition in the list of *Attribute Handles*. Characteristic Presentation Format declarations from other characteristic definitions may also be used.

A *Characteristic Aggregate Format* declaration may exist without a *Characteristic Presentation Format* declaration existing in the characteristic definition. The *Characteristic Aggregate Format* declaration may use *Characteristic Presentation Format* declarations from other characteristic definitions.

## 3.4 SUMMARY OF GATT PROFILE ATTRIBUTE TYPES

The following table summarizes the Attribute Types defined by the GATT Profile.

| Attribute Type | UUID | Description |
|---|---|---|
| «Primary Service» | 0x2800 | Primary Service Declaration |
| «Secondary Service» | 0x2801 | Secondary Service Declaration |
| «Include» | 0x2802 | Include Declaration |
| «Characteristic» | 0x2803 | Characteristic Declaration |
| «Characteristic Extended Properties» | 0x2900 | Characteristic Extended Properties |
| «Characteristic User Description» | 0x2901 | Characteristic User Description Descriptor |
| «Client Characteristic Configuration» | 0x2902 | Client Characteristic Configuration Descriptor |
| «Server Characteristic Configuration» | 0x2903 | Server Characteristic Configuration Descriptor |
| «Characteristic Format» | 0x2904 | Characteristic Format Descriptor |
| «Characteristic Aggregate Format» | 0x2905 | Characteristic Aggregate Format Descriptor |

*Table 3.18: Summary of GATT Profile Attribute types*

# 4 GATT FEATURE REQUIREMENTS

## 4.1 OVERVIEW

There are 11 features defined in the GATT Profile:

1. Server Configuration
2. Primary Service Discovery
3. Relationship Discovery
4. Characteristic Discovery
5. Characteristic Descriptor Discovery
6. Reading a Characteristic Value
7. Writing a Characteristic Value
8. Notification of a Characteristic Value
9. Indication of a Characteristic Value
10. Reading a Characteristic Descriptor
11. Writing a Characteristic Descriptor

Each of the features is mapped to procedures and sub-procedures. These procedures and sub-procedures describe how the Attribute Protocol is used to accomplish the corresponding feature.

## 4.2 FEATURE SUPPORT AND PROCEDURE MAPPING

The table below maps each feature to the procedures used for that feature, and indicates whether the procedure is optional or mandatory for that feature. The procedures are described in the referenced section.

| Item No. | Feature | Sub-Procedure | Ref. | Support in Client | Support in Server |
|---|---|---|---|---|---|
| 1 | Server Configuration | Exchange MTU | 4.3.1 | O | O |
| 2 | Primary Service Discovery | Discover All Primary Services | 4.4.1 | O | M |
| | | Discover Primary Services By Service UUID | 4.4.2 | O | M |
| 3 | Relationship Discovery | Find Included Services | 4.5.1 | O | M |

*Table 4.1: GATT feature mapping to procedures*

| Item No. | Feature | Sub-Procedure | Ref. | Support in Client | Support in Server |
|---|---|---|---|---|---|
| 4 | Characteristic Discovery | Discover All Characteristic of a Service | 4.6.1 | O | M |
| | | Discover Characteristic by UUID | 4.6.2 | O | M |
| 5 | Characteristic Descriptor Discovery | Discover All Characteristic Descriptors | 4.7.1 | O | M |
| 6 | Characteristic Value Read | Read Characteristic Value | 4.8.1 | O | M |
| | | Read Using Characteristic UUID | 4.8.1 | O | M |
| | | Read Long Characteristic Values | 4.8.2 | O | O |
| | | Read Multiple Characteristic Values | 4.8.3 | O | O |
| 7 | Characteristic Value Write | Write Without Response | 4.9.1 | O | C.1 |
| | | Signed Write Without Response | 4.9.2 | O | O |
| | | Write Characteristic Value | 4.9.3 | O | C.2 |
| | | Write Long Characteristic Values | 4.9.4 | O | O |
| | | Characteristic Value Reliable Writes | 4.9.5 | O | O |
| 8 | Characteristic Value Notification | Notifications | 4.10.1 | O | O |
| 9 | Characteristic Value Indication | Indications | 4.11.1 | M | C3 |
| 10 | Characteristic Descriptor Value Read | Read Characteristic Descriptors | 4.12.1 | O | O |
| | | Read Long Characteristic Descriptors | 4.12.2 | O | O |
| 11 | Characteristic Descriptor Value Write | Write Characteristic Descriptors | 4.12.3 | O | O |
| | | Write Long Characteristic Descriptors | 4.12.4 | O | O |

*Table 4.1: GATT feature mapping to procedures*

| Item No. | Feature | Sub-Procedure | Ref. | Support in Client | Support in Server |
|---|---|---|---|---|---|
| C1: Write Without Response is mandatory if Signed Write Without Response is supported otherwise optional<br>C2: Write Characteristic Value is mandatory if Write Long Characteristic Values is supported otherwise optional<br>C3: If *Service Change Characteristic* is present, this feature is mandatory, otherwise optional. | | | | | |

*Table 4.1: GATT feature mapping to procedures*

## 4.3 SERVER CONFIGURATION

This procedure is used by the client to configure the Attribute Protocol. This procedure has only one sub-procedure used to set the MTU sizes.

### 4.3.1 Exchange MTU

This sub-procedure is used by the client to set the ATT_MTU to the maximum possible value that can be supported by both devices when the client supports a value greater than the default ATT_MTU for the Attribute Protocol. This sub-procedure shall only be initiated once during a connection.

This sub-procedure shall not be used on a BR/EDR physical link since the MTU size is negotiated using L2CAP channel configuration procedures.

The Attribute Protocol *Exchange MTU Request* is used by this sub-procedure. The Client Rx MTU parameter shall be set to the maximum MTU that this client can receive.

Two possible responses can be sent from the server for the *Exchange MTU Request: Exchange MTU Response* and *Error Response*.

*Error Response* is returned if an error occurred on the server.

The server shall respond to this message with an *Exchange MTU Response* with the Server Rx MTU parameter set to the maximum MTU that this server can receive.

If the *Error Response* is sent by the server with the *Error Code* set to *Request Not Supported*, the *Attribute Opcode* is not supported and the default MTU shall be used.

Once the messages have been exchanged, the ATT_MTU shall be set to the minimum of the Client Rx MTU and Server Rx MTU values.

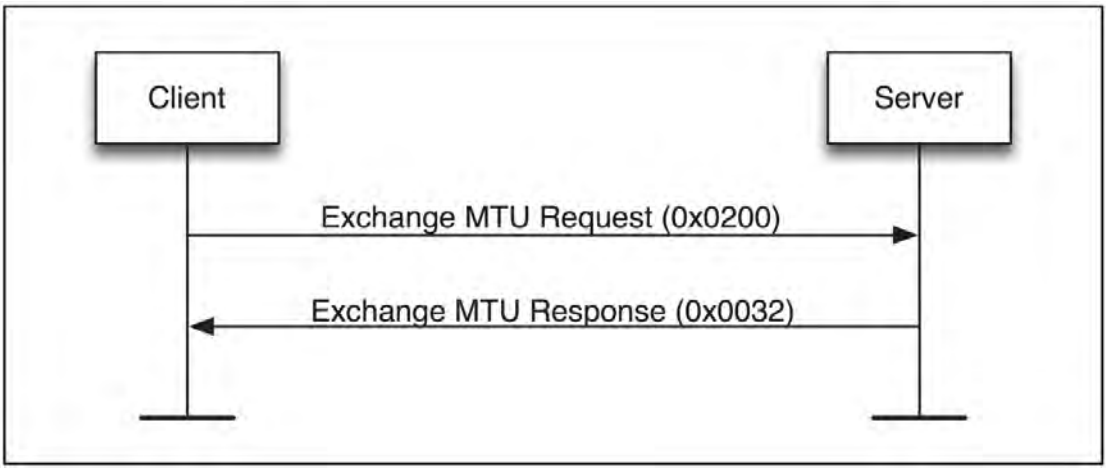



*Figure 4.1: Exchange MTU*

For example, in Figure 4.1, based on the exchanged ATT_MTU value in the ATT_MTU would be 0x0032.

## 4.4 PRIMARY SERVICE DISCOVERY

This procedure is used by a client to discover primary services on a server. Once the primary services are discovered, additional information about the primary services can be accessed using other procedures, including characteristic discovery and relationship discovery to find other related primary and secondary services.

There are two sub-procedures that can be used for primary service discovery: Discover All Primary Services and Discover Primary Services by Service UUID.

GATT service discovery over BR/EDR transport will list services that only run over an LE transport and therefore only SDP service discovery shall be used on BR/EDR.

### 4.4.1 Discover All Primary Services

This sub-procedure is used by a client to discover all the primary services on a server.

The Attribute Protocol *Read By Group Type Request* shall be used with the Attribute Type parameter set to the UUID for «Primary Service». The *Starting Handle* shall be set to 0x0001 and the *Ending Handle* shall be set to 0xFFFF.

Two possible responses can be sent from the server for the *Read By Group Type Request: Read By Group Type Response and Error Response*.

*Error Response* is returned if an error occurred on the server.

*Read By Group Type Response* returns a list of *Attribute Handle*, *End Group Handle,* and *Attribute Value* tuples corresponding to the services supported by the server. Each *Attribute Value* contained in the response is the Service UUID of a service supported by the server. The *Attribute Handle* is the handle for the service declaration. The End Group Handle is the handle of the last attribute within the service definition. The *Read By Group Type Request* shall be called again with the *Starting Handle* set to one greater than the last *End Group Handle* in the *Read By Group Type Response*.

The sub-procedure is complete when the *Error Response* is received and the *Error Code* is set to *Attribute Not Found*.

It is permitted to end the sub-procedure early if a desired primary service is found prior to discovering all the primary services on the server.

Note: The service declaration described in Section 3.1 specifies that the service declaration is readable and requires no authentication or authorization, therefore insufficient authentication or read not permitted errors shall not occur.

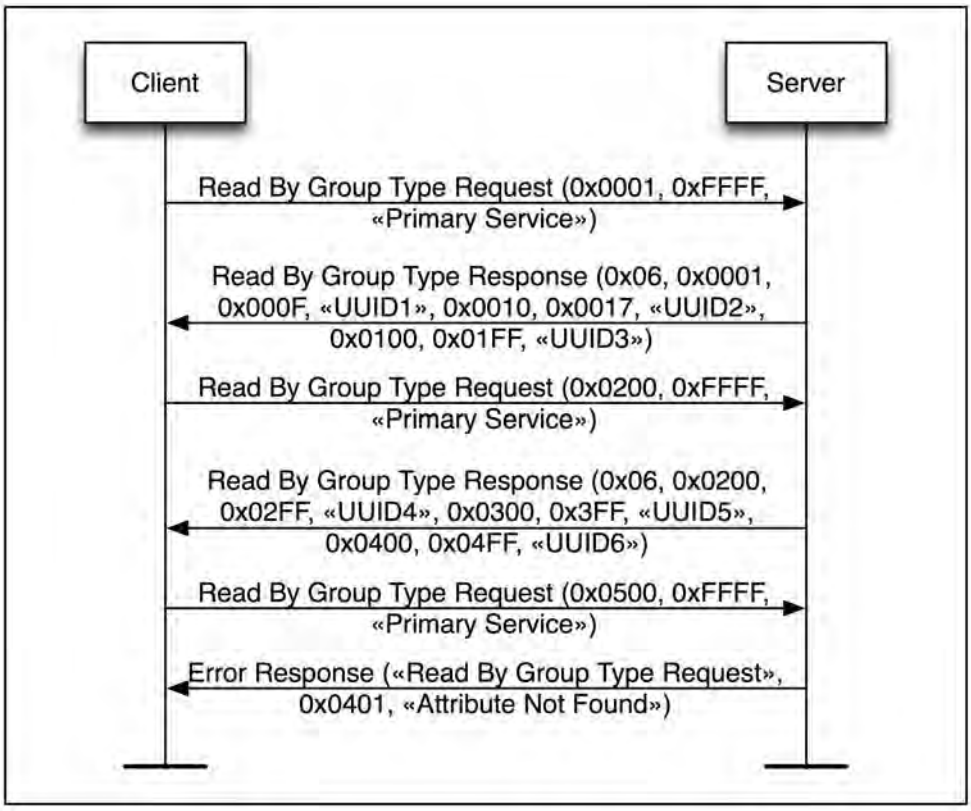



*Figure 4.2: Discover All Primary Services example*

### 4.4.2 Discover Primary Service by Service UUID

This sub-procedure is used by a client to discover a specific primary service on a server when only the Service UUID is known. The specific primary service

may exist multiple times on a server. The primary service being discovered is identified by the service UUID.

The Attribute Protocol *Find By Type Value Request* shall be used with the Attribute Type parameter set to the UUID for «Primary Service» and the *Attribute Value* set to the 16-bit Bluetooth UUID or 128-bit UUID for the specific primary service. The *Starting Handle* shall be set to 0x0001 and the *Ending Handle* shall be set to 0xFFFF.

Two possible responses can be sent from the server for the *Find By Type Value Request: Find By Type Value Response and Error Response*.

*Error Response* is returned if an error occurred on the server.

*Find By Type Value Response* returns a list of *Attribute Handle* ranges. The *Attribute Handle* range is the starting handle and the ending handle of the service definition. If the *Attribute Handle* range for the Service UUID being searched is returned and the End Found Handle is not 0xFFFF, the *Find By Type Value Request* may be called again with the *Starting Handle* set to one greater than the last *Attribute Handle* range in the Find By Type Value Response.

The sub-procedure is complete when the *Error Response* is received and the *Error Code* is set to *Attribute Not Found*.

It is permitted to end the sub-procedure early if a desired primary service is found prior to discovering all the primary services of the specified service UUID supported on the server.

Note: The service declaration described in Section 3.1 specifies that the service declaration is readable and requires no authentication or authorization, therefore insufficient authentication or read not permitted errors shall not occur.

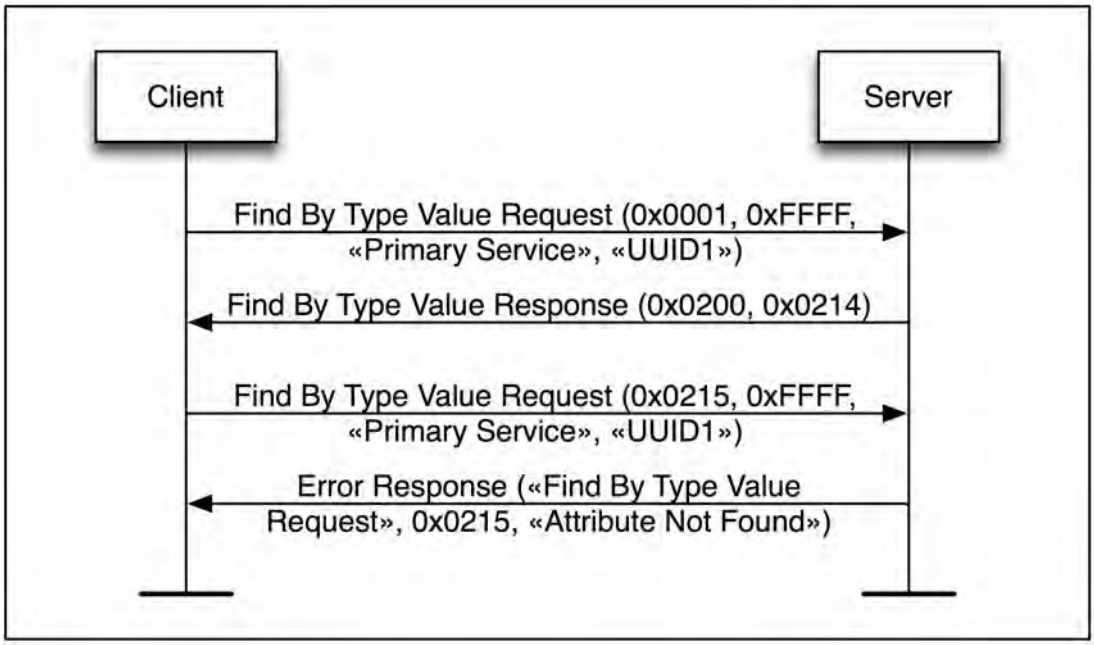



*Figure 4.3: Discover Primary Service by Service UUID example*

## 4.5 RELATIONSHIP DISCOVERY

This procedure is used by a client to discover service relationships to other services.

There is one sub-procedure that can be used for relationship discovery: Find Included Services.

### 4.5.1 Find Included Services

This sub-procedure is used by a client to find include service declarations within a service definition on a server. The service specified is identified by the service handle range.

The Attribute Protocol Read *By Type Request* shall be used with the *Attribute Type* parameter set to the UUID for «Include» The *Starting Handle* shall be set to starting handle of the specified service and the *Ending Handl*e shall be set to the ending handle of the specified service. It is permitted to end the sub-procedure early if a desired included service is found prior to discovering all the included services of the specified service supported on the server.

Two possible responses can be sent from the server for the *Read By Type Request: Read By Type Response and Error Response*.

*Error Response* is returned if an error occurred on the server.

*Read By Type Response* returns a set of *Attribute Handle* and *Attribute Value* pairs corresponding to the included services in the service definition. Each

*Attribute Value* contained in the response is composed of the *Attribute Handle* of the included service declaration and the *End Group Handle*. If the service UUID is a 16-bit Bluetooth UUID it is also returned in the response.The *Read By Type Request* shall be called again with the *Starting Handle* set to one greater than the last *Attribute Handle* in the *Read By Type Response*.

The sub-procedure is complete when either the *Error Response* is received with the *Error Code* set to *Attribute Not Found* or the *Read By Type Response* has an *Attribute Handle* of the included service declaration that is equal to the *Ending Handle* of the request.

To get the included service UUID when the included service uses a 128-bit UUID, the *Read Request* is used. The *Attribute Handle* for the *Read Request* is the *Attribute Handle* of the included service.

Note: The include declaration described in Section 3.2 specifies that the include declaration is readable and requires no authentication or authorization, therefore insufficient authentication or read not permitted errors shall not occur.

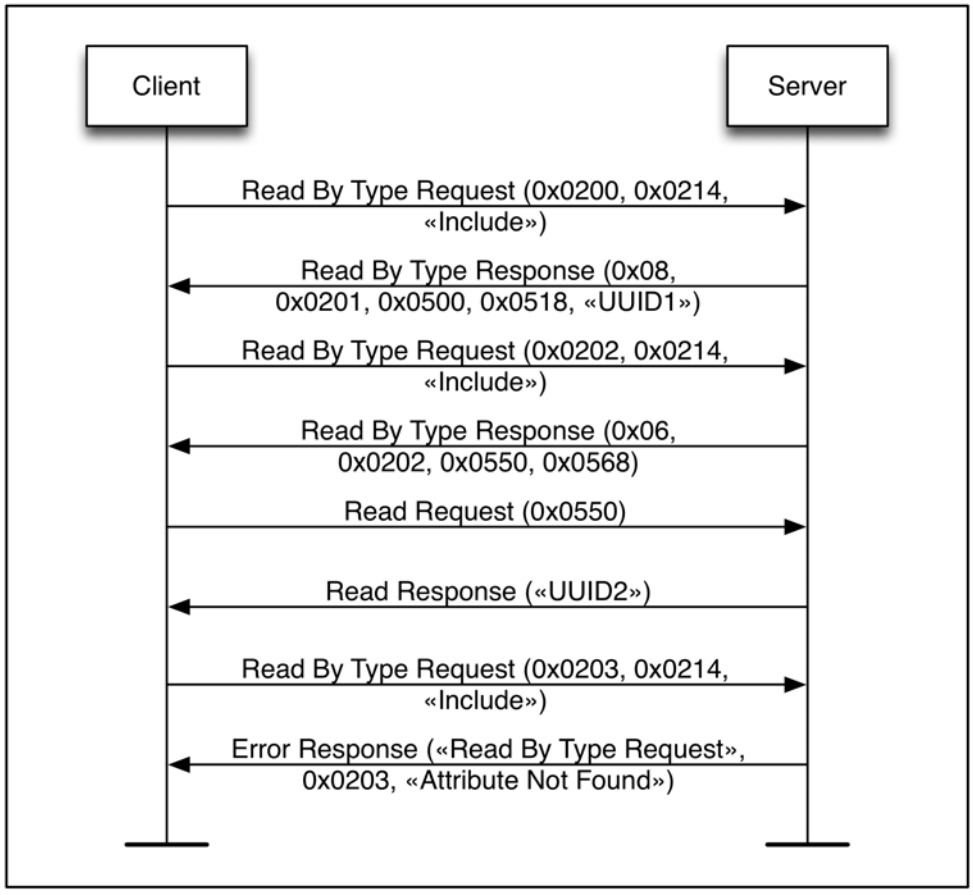



*Figure 4.4: Find Included Services example*

## 4.6 CHARACTERISTIC DISCOVERY

This procedure is used by a client to discover service characteristics on a server. Once the characteristics are discovered additional information about the characteristics can be discovered or accessed using other procedures.

There are two sub-procedures that can be used for characteristic discovery: Discover All Characteristics of a Service and Discover Characteristics by UUID.

### 4.6.1 Discover All Characteristics of a Service

This sub-procedure is used by a client to find all the characteristic declarations within a service definition on a server when only the service handle range is known. The service specified is identified by the service handle range.

The Attribute Protocol *Read By Type Request* shall be used with the *Attribute Type* parameter set to the UUID for «Characteristic» The *Starting Handle* shall be set to starting handle of the specified service and the *Ending Handle* shall be set to the ending handle of the specified service.

Two possible responses can be sent from the server for the *Read By Type Request: Read By Type Response* and *Error Response*.

*Error Response* is returned if an error occurred on the server.

*Read By Type Response* returns a list of *Attribute Handle* and *Attribute Value* pairs corresponding to the characteristics in the service definition. The *Attribute Handle* is the handle for the characteristic declaration. The *Attribute Value* is the Characteristic Properties, Characteristic Value Handle and Characteristic UUID. The Read By Type Request shall be called again with the *Starting Handle* set to one greater than the last *Attribute Handle* in the *Read By Type Response*.

The sub-procedure is complete when the *Error Response* is received and the *Error Code* is set to *Attribute Not Found* or the *Read By Type Response* has an *Attribute Handle* that is equal to the *Ending Handle* of the request.

It is permitted to end the sub-procedure early if a desired characteristic is found prior to discovering all the characteristics of the specified service supported on the server.

Note: The characteristic declaration described in Section 3.3 specifies that the characteristic declaration is readable and requires no authentication or authorization, therefore insufficient authentication or read not permitted errors should not occur.

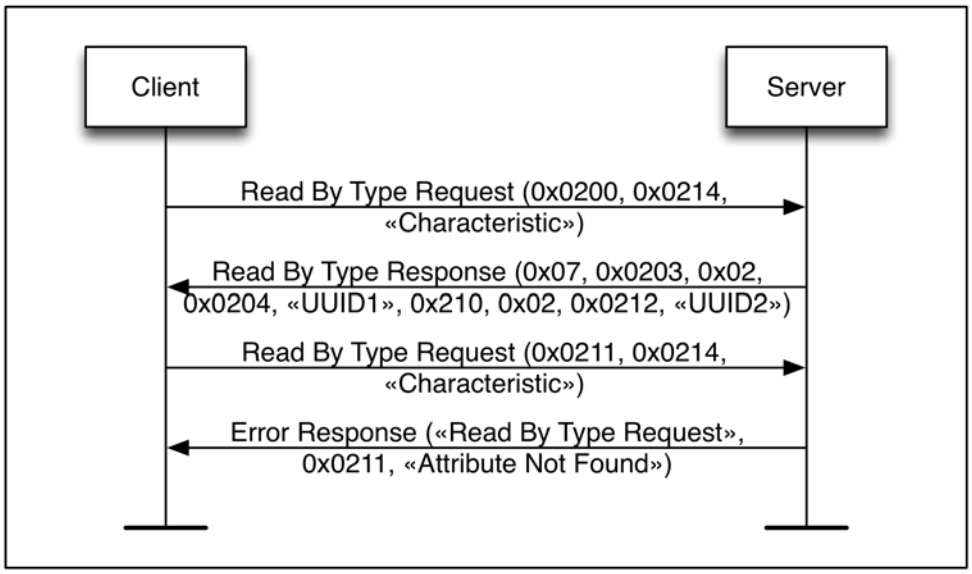



*Figure 4.5: Discover All Characteristics of a Service example*

### 4.6.2 Discover Characteristics by UUID

This sub-procedure is used by a client to discover service characteristics on a server when only the service handle ranges are known and the characteristic UUID is known. The specific service may exist multiple times on a server. The characteristic being discovered is identified by the characteristic UUID.

The Attribute Protocol *Read By Type Request* is used to perform the beginning of the sub-procedure. The *Attribute Type* is set to the UUID for «Characteristic» and the *Starting Handle* and *Ending Handle* parameters shall be set to the service handle range.

Two possible responses can be sent from the server for the *Read By Type Request: Read By Type Response* and *Error Response*.

*Error Response* is returned if an error occurred on the server.

*Read By Type Response* returns a list of *Attribute Handle* and *Attribute Value* pairs corresponding to the characteristics contained in the handle range provided. Each *Attribute Value* in the list is the *Attribute Value* for the characteristic declaration. The *Attribute Value* contains the characteristic properties, Characteristic *Value Handle* and characteristic UUID. The *Attribute Value* for each *Attribute Handle* and *Attribute Value* pairs are checked for a matching characteristic UUID. Once found, the sub-procedure continues until the end of the service handle range is exhausted. The *Read By Type Request* is called again with the *Starting Handle* set to one greater than the last *Attribute Handle* in the *Read By Type Response*.

If the *Error Response* is sent by the server with the *Error Code* set to *Attribute Not Found*, the characteristic does not exist on the server within the handle range provided.

It is permitted to end the sub-procedure early if a desired characteristic is found prior to discovering all the characteristics for the specified service supported on the server.

Note: The characteristic declaration described in Section 3.3 specifies that the characteristic declaration is readable and requires no authentication or authorization, therefore insufficient authentication or read not permitted errors shall not occur.

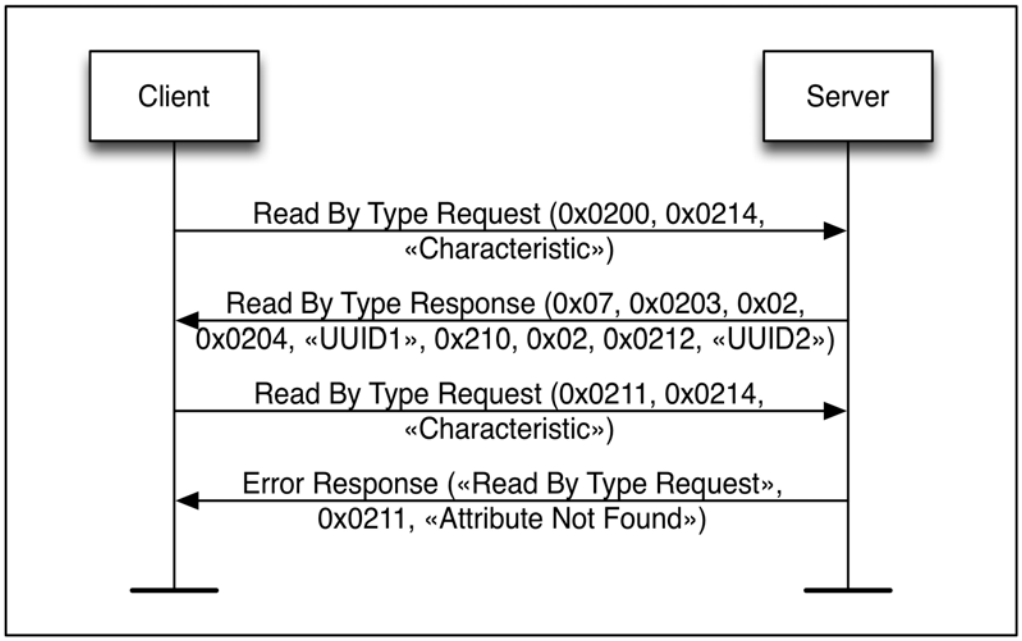



*Figure 4.6: Discover Characteristics by UUID example*

## 4.7 CHARACTERISTIC DESCRIPTOR DISCOVERY

This procedure is used by a client to discover characteristic descriptors of a characteristic. Once the characteristic descriptors are discovered additional information about the characteristic descriptors can be accessed using other procedures.

There is one sub-procedure that can be used for characteristic descriptor discovery: Discover All Characteristic Descriptors.

### 4.7.1 Discover All Characteristic Descriptors

This sub-procedure is used by a client to find all the characteristic descriptor's Attribute Handles and Attribute Types within a characteristic definition when

only the characteristic handle range is known. The characteristic specified is identified by the characteristic handle range.

The Attribute Protocol *Find Information Request* shall be used with the Starting Handle set to the handle of the specified characteristic value + 1 and the *Ending Handle* set to the ending handle of the specified characteristic.

Two possible responses can be sent from the server for the *Find Information Request: Find Information Response* and *Error Response*.

Error Response is returned if an error occurred on the server.

*Find Information Response* returns a list of *Attribute Handle* and *Attribute Value* pairs corresponding to the characteristic descriptors in the characteristic definition. The *Attribute Handle* is the handle for the characteristic descriptor declaration. The *Attribute Value* is the Characteristic Descriptor UUID. The *Find Information Request* shall be called again with the *Starting Handle* set to one greater than the last *Attribute Handle* in the Find Information Response.

The sub-procedure is complete when the *Error Response* is received and the *Error Code* is set to *Attribute Not Found* or the *Find Information* Response has an *Attribute Handle* that is equal to the *Ending Handle* of the request.

It is permitted to end the sub-procedure early if a desired Characteristic Descriptor is found prior to discovering all the characteristic descriptors of the specified characteristic.

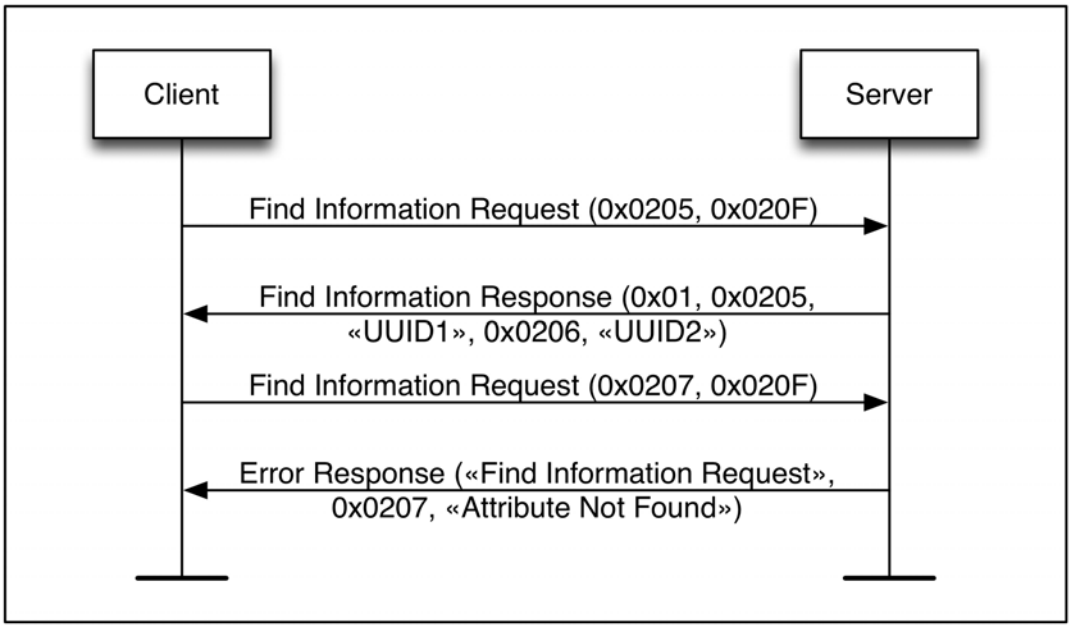



*Figure 4.7: Discover All Characteristic Descriptors example*

## 4.8 CHARACTERISTIC VALUE READ

This procedure is used to read a *Characteristic Value* from a server. There are four sub-procedures that can be used to read a *Characteristic Value*: Read Characteristic Value, Read Using Characteristic UUID, Read Long Characteristic Values, and Read Multiple Characteristic Values.

### 4.8.1 Read Characteristic Value

This sub-procedure is used to read a *Characteristic Value* from a server when the client knows the *Characteristic Value Handle*. The Attribute Protocol *Read Request* is used with the *Attribute Handle* parameter set to the *Characteristic Value Handle*. The Read Response returns the *Characteristic Value* in the *Attribute Value* parameter.

The *Read Response* only contains a *Characteristic Value* that is less than or equal to (ATT_MTU – 1) octets in length. If the *Characteristic Value* is greater than (ATT_MTU – 1) octets in length, the Read Long Characteristic Value procedure may be used if the rest of the *Characteristic Value* is required.

An *Error Response* shall be sent by the server in response to the *Read Request* if insufficient authentication, insufficient authorization, insufficient encryption key size is used by the client, or if a read operation is not permitted on the *Characteristic Value*. The *Error Code* parameter is set as specified in the Attribute Protocol.

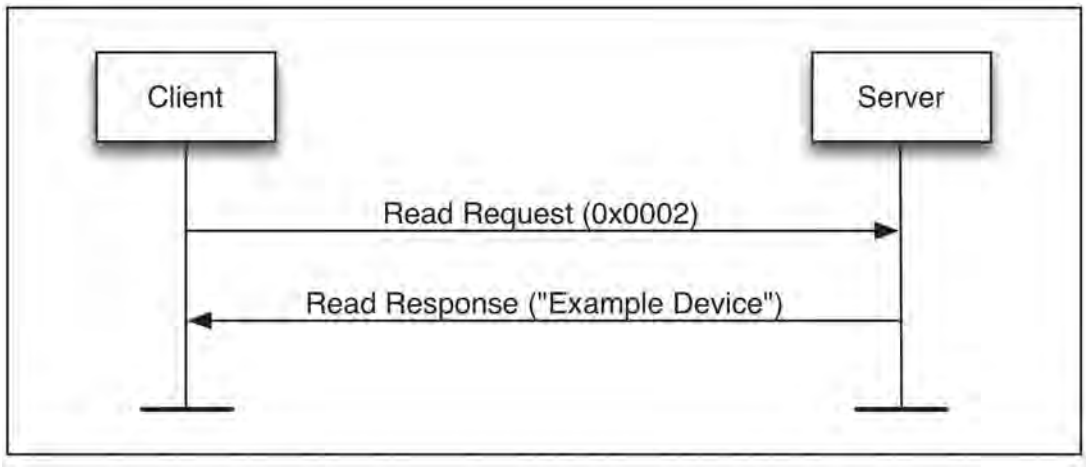



*Figure 4.8: Read Characteristic Value example*

### 4.8.2 Read Using Characteristic UUID

This sub-procedure is used to read a *Characteristic Value* from a server when the client only knows the characteristic UUID and does not know the handle of the characteristic.

The Attribute Protocol *Read By Type Request* is used to perform the sub-procedure. The Attribute Type is set to the known characteristic UUID and the Starting Handle and Ending Handle parameters shall be set to the range over which this read is to be performed. This is typically the handle range for the service in which the characteristic belongs.

Two possible responses can be sent from the server for the *Read By Type Request: Read By Type Response* and *Error Response*.

Error Response is returned if an error occurred on the server.

*Read By Type Response* returns a list of *Attribute Handle* and *Attribute Value* pairs corresponding to the characteristics contained in the handle range provided.

If the Error Response is sent by the server with the *Error Code* set to *Attribute Not Found*, the characteristic does not exist on the server within the handle range provided.

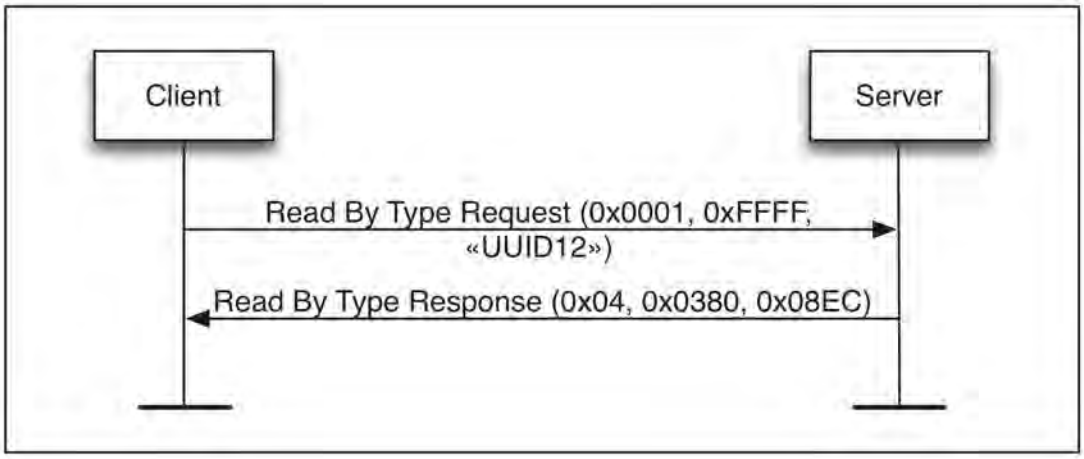



*Figure 4.9: Read Using Characteristic UUID example*

### 4.8.3 Read Long Characteristic Values

This sub-procedure is used to read a *Characteristic Value* from a server when the client knows the *Characteristic Value Handle* and the length of the *Characteristic Value* is longer than can be sent in a single *Read Response* Attribute Protocol message.

The Attribute Protocol *Read Blob Request* is used to perform this sub-procedure. The *Attribute Handle* shall be set to the *Characteristic Value Handle* of the *Characteristic Value* to be read. The Value Offset parameter shall be the offset within the *Characteristic Value* to be read. To read the complete *Characteristic Value* the offset should be set to 0x00 for the first *Read Blob Request*. The offset for subsequent *Read Blob Requests* is the next octet that has yet to

be read. The *Read Blob Request* is repeated until the *Read Blob Response's Part Attribute Value* parameter is shorter than (ATT_MTU-1).

For each *Read Blob Request* a *Read Blob Response* is received with a portion of the Characteristic Value contained in the *Part Attribute Value* parameter.

An *Error Response* shall be sent by the server in response to the *Read Blob Request* if insufficient authentication, insufficient authorization, insufficient encryption key size is used by the client, or if a read operation is not permitted on the *Characteristic Value*. The Error Code parameter is set as specified in the Attribute Protocol. If the *Characteristic Value* is not longer than (ATT_MTU – 1) an *Error Response* with the *Error Code* set to Attribute Not Long shall be received on the first *Read Blob Reques*t.

Note: The *Read Blob Request* may be used to read the remainder of an Attribute where the first part was read using a simple *Read Request*.

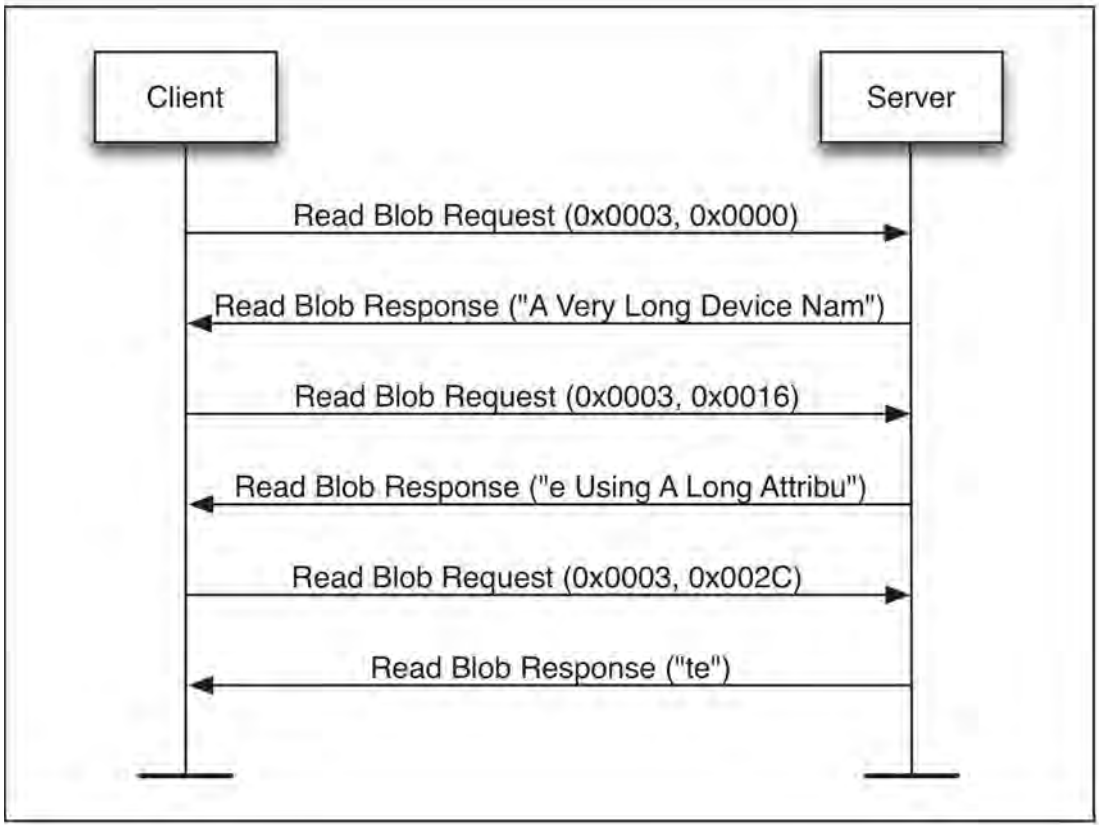



*Figure 4.10: Read Long Characteristic Values example*

### 4.8.4 Read Multiple Characteristic Values

This sub-procedure is used to read multiple *Characteristic Values* from a server when the client knows the *Characteristic Value Handles*. The Attribute Protocol *Read Multiple Requests* is used with the *Set Of Handles* parameter set to the *Characteristic Value Handles*. The *Read Multiple Response* returns the *Characteristic Values* in the *Set Of Values* parameter.

The *Read Multiple Response* only contains a set of *Characteristic Values* that is less than or equal to (ATT_MTU – 1) octets in length. If the *Set Of Values* is greater than (ATT_MTU – 1) octets in length, only the first (ATT_MTU – 1) octets are included in the response.

Note: A client should not request multiple *Characteristic Values* when the response's *Set Of Values* parameter is equal to (ATT_MTU – 1) octets in length since it is not possible to determine if the last *Characteristic Value* was read or additional *Characteristic Values* exist but were truncated.

An *Error Response* shall be sent by the server in response to the *Read Multiple Request* if insufficient authentication, insufficient authorization, insufficient encryption key size is used by the client, or if a read operation is not permitted on any of the *Characteristic Values*. The *Error Code* parameter is set as specified in the Attribute Protocol.

Refer to the Attribute Protocol specification for the format of the *Set Of Handles* and *Set Of Values* parameter.

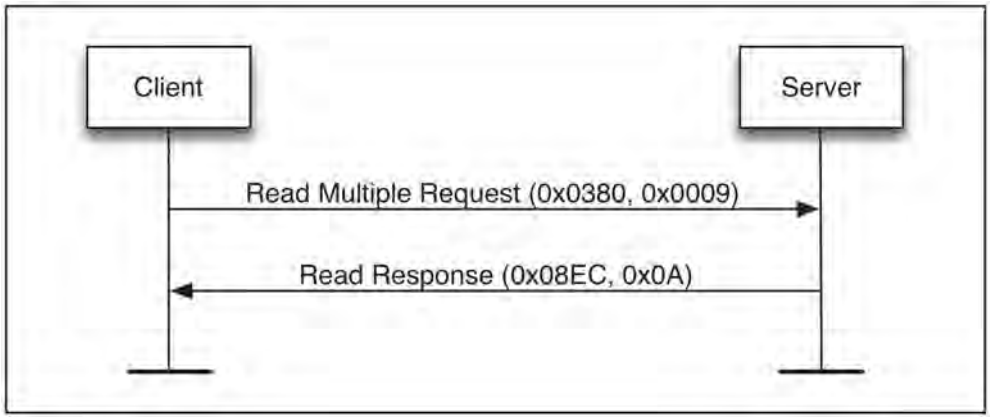



*Figure 4.11: Read Multiple Characteristic Values example*

## 4.9 CHARACTERISTIC VALUE WRITE

This procedure is used to write a *Characteristic Value* to a server.

There are five sub-procedures that can be used to write a *Characteristic Value*: Write Without Response, Signed Write Without Response, Write Characteristic Value, Write Long Characteristic Values and Reliable Writes.

### 4.9.1 Write Without Response

This sub-procedure is used to write a *Characteristic Value* to a server when the client knows the *Characteristic Value Handle* and the client does not need an acknowledgement that the write was successfully performed. This sub-procedure only writes the first (ATT_MTU – 3) octets of a *Characteristic Value*. This sub-procedure cannot be used to write a long characteristic; instead the *Write Long Characteristic Values* sub-procedure should be used.

The Attribute Protocol *Write Command* is used for this sub-procedure. The *Attribute Handle* parameter shall be set to the *Characteristic Value Handle*. The *Attribute Value* parameter shall be set to the new *Characteristic Value*.

If the *Characteristic Value* write request is the wrong size, or has an invalid value as defined by the profile, then the write shall not succeed and no error shall be generated by the server.

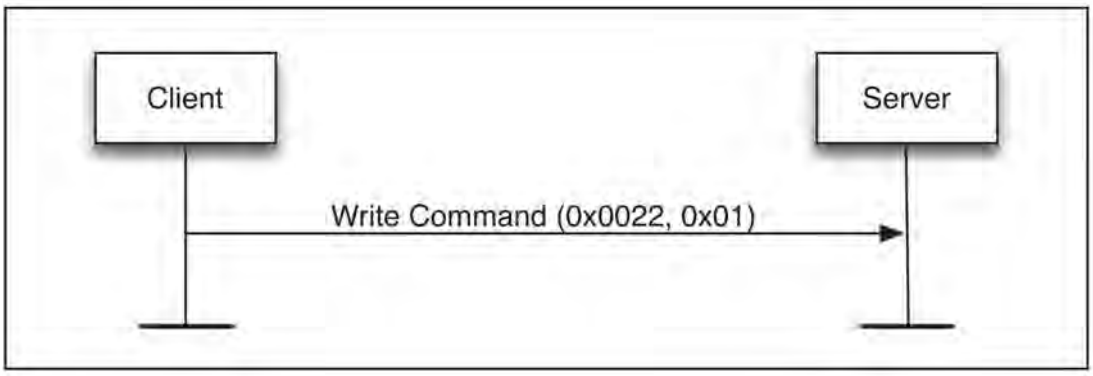



*Figure 4.12: Write Without Response example*

### 4.9.2 Signed Write Without Response

This sub-procedure is used to write a *Characteristic Value* to a server when the client knows the *Characteristic Value Handle* and the ATT Bearer is not encrypted. This sub-procedure shall only be used if the *Characteristic Properties* authenticated bit is enabled and the client and server device share a bond as defined in [Vol. 3] Part C, Generic Access Profile.

This sub-procedure only writes the first (ATT_MTU – 15) octets of an *Attribute Value.* This sub-procedure cannot be used to write a long Attribute.

The Attribute Protocol *Write Command* is used for this sub-procedure. The *Attribute Handle* parameter shall be set to the *Characteristic Value Handle*. The *Attribute Value* parameter shall be set to the new *Characteristic Value* authenticated by signing the value, as defined in the Security Manager [Vol. 3] Part H, Section 2.4.5.

If the authenticated *Characteristic Value* that is written is the wrong size, has an invalid value as defined by the profile, or the signed value does not authenticate the client, then the write shall not succeed and no error shall be generated by the server.

Note: On BR/EDR, the ATT Bearer is always encrypted, due to the use of Security Mode 4, therefore this sub-procedure shall not be used.

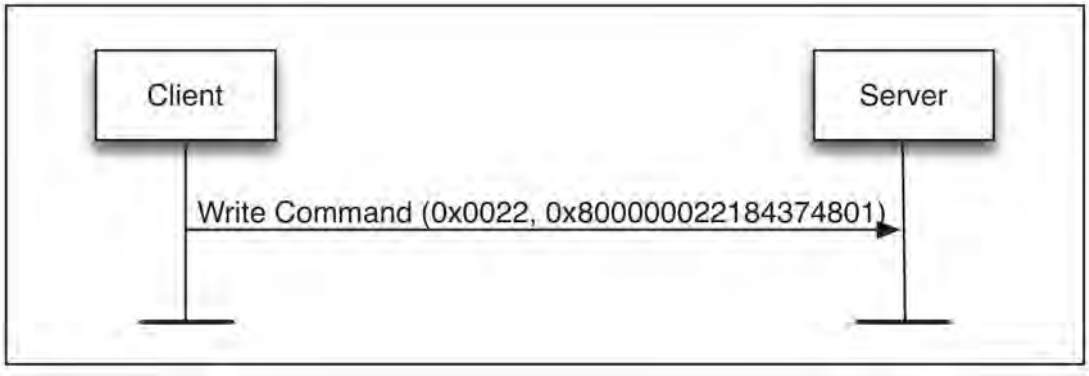



*Figure 4.13: Signed Write Without Response example*

### 4.9.3 Write Characteristic Value

This sub-procedure is used to write a *Characteristic Value* to a server when the client knows the *Characteristic Value Handle*. This sub-procedure only writes the first (ATT_MTU – 3) octets of a *Characteristic Value*. This sub-procedure cannot be used to write a long Attribute; instead the *Write Long Characteristic Values* sub-procedure should be used.

The Attribute Protocol *Write Request* is used to for this sub-procedure. The *Attribute Handle* parameter shall be set to the *Characteristic Value Handle*. The *Attribute Value* parameter shall be set to the new characteristic.

A *Write Response* shall be sent by the server if the write of the *Characteristic Value* succeeded.

An *Error Response* shall be sent by the server in response to the *Write Request* if insufficient authentication, insufficient authorization, insufficient encryption key size is used by the client, or if a write operation is not permitted on the *Characteristic Value*. The *Error Code* parameter is set as specified in the Attribute Protocol. If the *Characteristic Value* that is written is the wrong size, or has an invalid value as defined by the profile, then the value shall not be written and an *Error Response* shall be sent with the *Error Code* set to *Application Error* by the server.

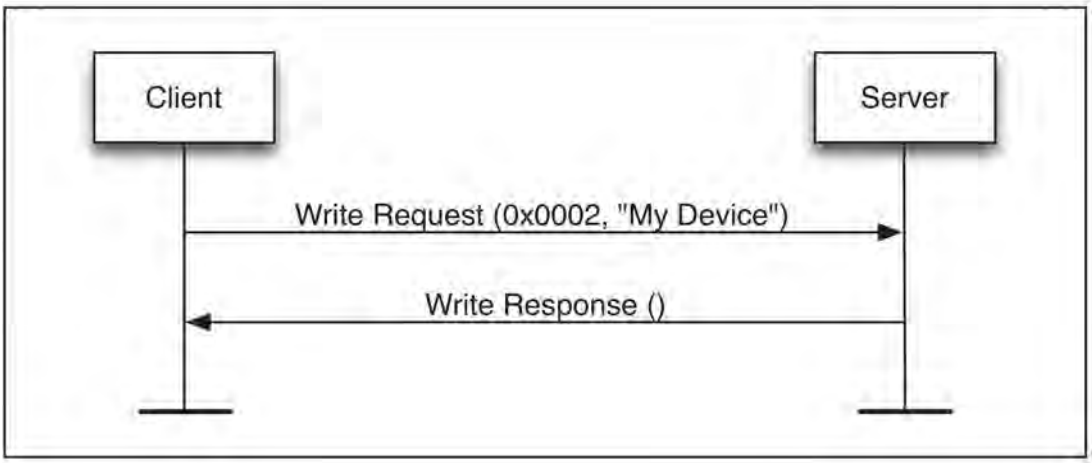



*Figure 4.14: Write Characteristic Value example*

### 4.9.4 Write Long Characteristic Values

This sub-procedure is used to write a *Characteristic Value* to a server when the client knows the *Characteristic Value Handle* but the length of the *Characteristic Value* is longer than can be sent in a single *Write Request* Attribute Protocol message.

The Attribute Protocol *Prepare Write Request* and *Execute Write Request* are used to perform this sub-procedure. The *Attribute Handle* parameter shall be set to the *Characteristic Value Handle* of the *Characteristic Value* to be written. The *Part Attribute Value* parameter shall be set to the part of the *Attribute Value* that is being written. The *Value Offset* parameter shall be the offset within the *Characteristic Value* to be written. To write the complete *Characteristic Value* the offset should be set to 0x0000 for the first *Prepare Write Request.* The offset for subsequent *Prepare Write Request*s is the next octet that has yet to be written. The *Prepare Write Request* is repeated until the complete *Characteristic Value* has been transferred, after which an Executive Write Request is used to write the complete value.

Note: The values in the Prepare Write Response do not need to be verified in this sub-procedure.

An *Error Response* shall be sent by the server in response to the *Prepare Write Request* if insufficient authentication, insufficient authorization, insufficient encryption key size is used by the client, or if a write operation is not permitted on the *Characteristic Value*. The *Error Code* parameter is set as specified in the Attribute Protocol. If the *Attribute Value* that is written is the wrong size, or has an invalid value as defined by the profile, then the write shall not succeed and an *Error Response* shall be sent with the *Error Code* set to *Application Error* by the server.

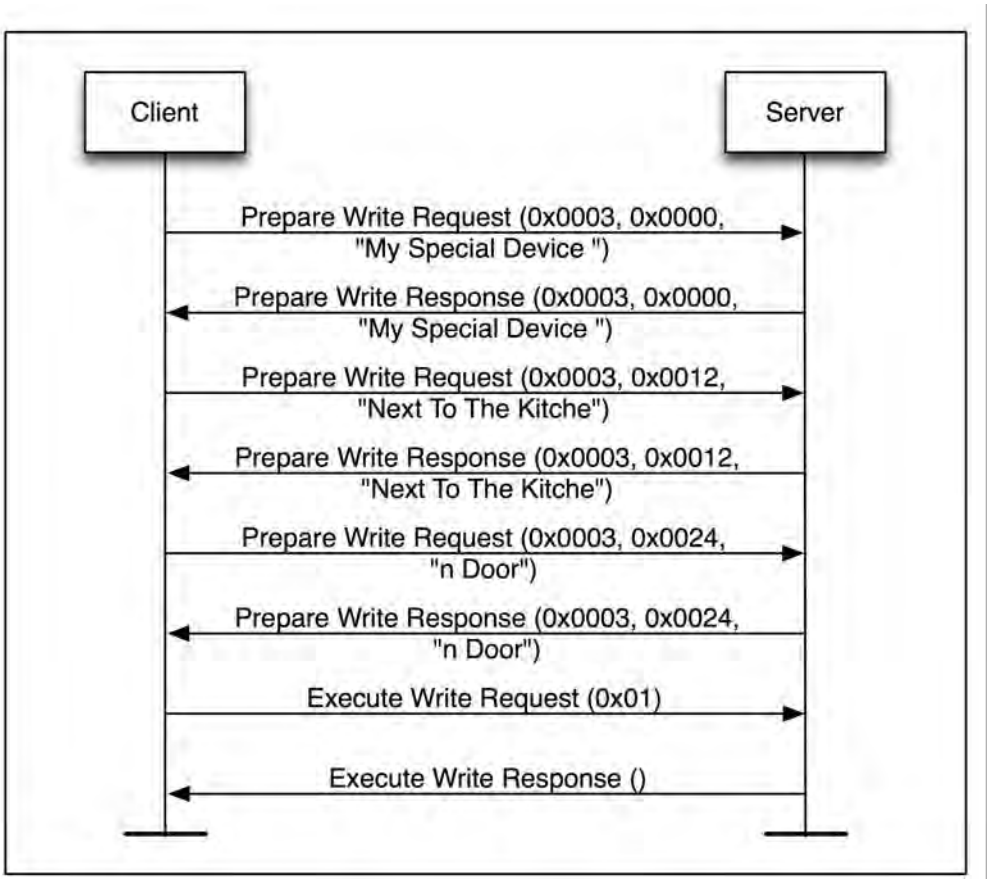



*Figure 4.15: Write Long Characteristic Values example*

### 4.9.5 Reliable Writes

This sub-procedure is used to write a *Characteristic Value* to a server when the client knows the *Characteristic Value Handle*, and assurance is required that the correct *Characteristic Value* is going to be written by transferring the *Characteristic Value* to be written in both directions before the write is performed. This sub-procedure can also be used when multiple values must be written, in order, in a single operation.

The sub-procedure has two phases; the first phase prepares the *Characteristic Values* to be written. To do this, the client transfers the *Characteristic Values* to the server. The server checks the validity of the *Characteristic Values*. The client also checks each *Characteristic Value* to verify it was correctly received by the server using the server responses. Once this is complete, the second *phase performs the execution of all of the prepared* Characteristic Value writes on the server from this client.

In the first phase, the Attribute Protocol *Prepare Write Request* is used. The *Attribute Handle* shall be set to the *Characteristic Value Handle* that is to be prepared to write. The *Value Offset* and *Part Attribute Value* parameter shall be set to the new *Characteristic Value*.

There are two possible responses; *Prepare Write Response* or *Error Response*.

If the number of prepared write requests exceeds the number of prepared writes supported, then an *Error Response* with the *Error Code* set to *Prepare Queue Full* shall be sent by the server.

An *Error Response* shall be sent by the server in response to the *Prepare Write Request* if insufficient authentication, insufficient authorization, insufficient encryption key size is used by the client, or if a write operation is not permitted on the *Characteristic Value*. The *Error Code* parameter is set as specified in the Attribute Protocol. If the *Characteristic Value* that is written is the wrong size, or has an invalid value as defined by the profile, then the write shall not succeed, and an *Error Response* with the *Error Code* set to *Application Error* shall be sent by the server.

If a *Characteristic Value* is prepared two or more times during this sub-procedure, then all prepared values are written multiple times to the same *Characteristic Value* in the order that they were prepared.

If a *Prepare Write Response* is returned, then the *Value Offset* and *Part Attribute Value* parameter in the response shall be checked with the *Value Offset* and *Part Attribute Value* parameter that was sent in the *Prepare Write Request;* if they are different, then the value has been corrupted during transmission, and the sub-procedure shall be aborted by sending an *Execute Write Request* with the Flags parameter set to 0x00 to cancel all prepared writes. The complete sub-procedure may be restarted.

*Multiple Prepare Write Requests* can be sent by a client, each of which will be queued by the server.

In the second phase, the Attribute Protocol *Execute Write Request* is used. The Attribute Flags parameter shall be set to 0x01 to immediately write all pending prepared values in the order that they were prepared. The server shall write the prepared writes once it receives this request and shall only send the *Execute Write Response* once all the prepared values have been successfully written. If an application error occurs while writing these Attributes the server shall instead send the *Error Response* with an *Error Code* set to *Application Error* by the server. The state of the Characteristic Values that were prepared is undefined.

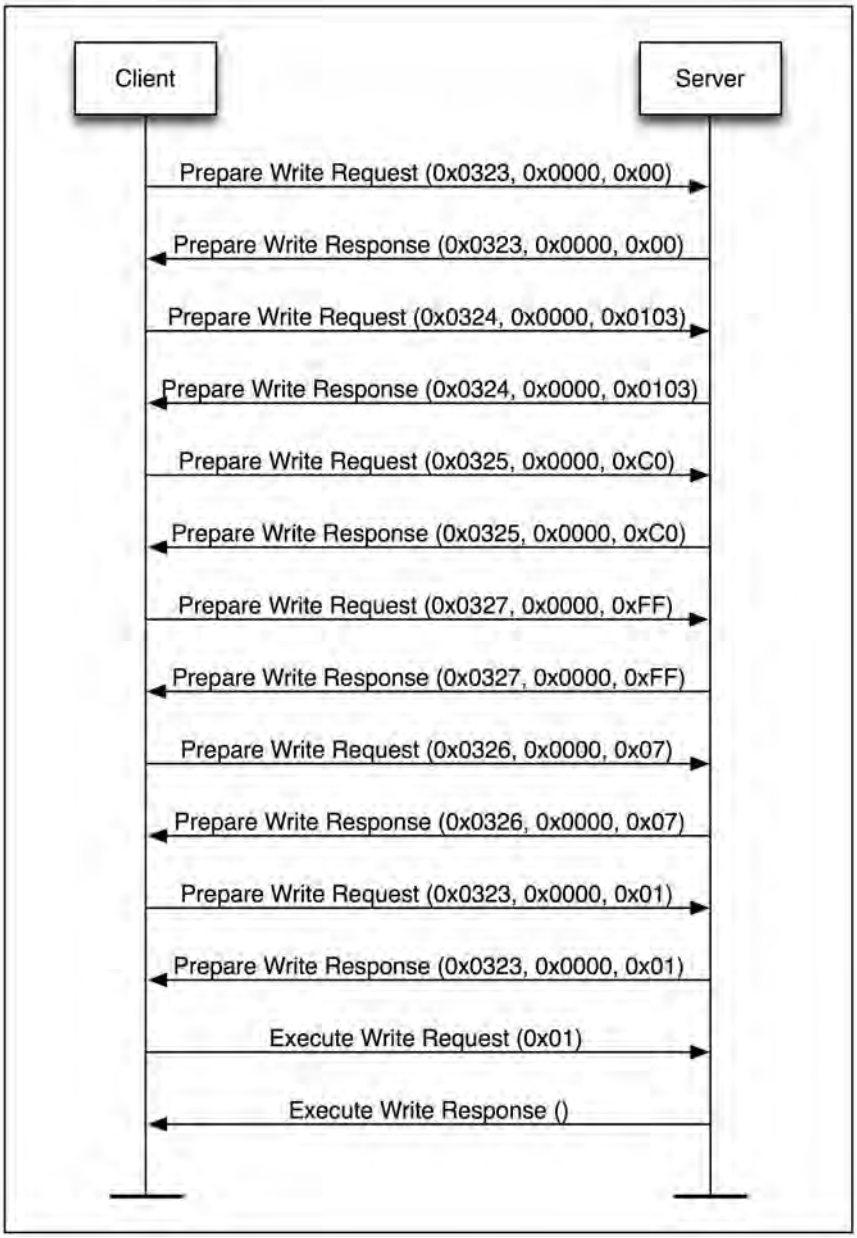



*Figure 4.16: Reliable Writes example[*

## 4.10 CHARACTERISTIC VALUE NOTIFICATION

This procedure is used to notify a client of the value of a *Characteristic Value* from a server. There is one sub-procedure that can be used to notify a value: Notifications. Notifications can be configured using the Characteristic Configuration descriptor (See Section 3.3.3.3).

A profile defines when to use Notifications.

### 4.10.1 Notifications

This sub-procedure is used when a server is configured to notify a *Characteristic Value* to a client without expecting any Attribute Protocol layer acknowledgement that the notification was successfully received.

The Attribute Protocol *Handle Value Notification* is used to perform this sub-procedure. The Attribute Handle parameter shall be set to the *Characteristic Value Handle* being notified, and the *Attribute Value* parameter shall be set to the *Characteristic Value*.




*Figure 4.17: Notifications example*

## 4.11 CHARACTERISTIC VALUE INDICATIONS

This procedure is used to indicate the *Characteristic Value* from a server to a client. There is one sub-procedure that can be used to indicate a value: Indications. Indications can be configured using the Characteristic Configuration (See Section 3.3.3.3).

A profile defines when to use Indications.

### 4.11.1 Indications

This sub-procedure is used when a server is configured to indicate a *Characteristic Value* to a client and expects an Attribute Protocol layer acknowledgement that the indication was successfully received.

The Attribute Protocol *Handle Value Indication* is used to perform this sub-procedure. The *Attribute Handle* parameter shall be set to the *Characteristic Value Handle* being indicated, and the *Attribute Value* parameter shall be set to the characteristic. Once the *Handle Value Indication* is received by the client, the client shall respond with a *Handle Value Confirmation*.

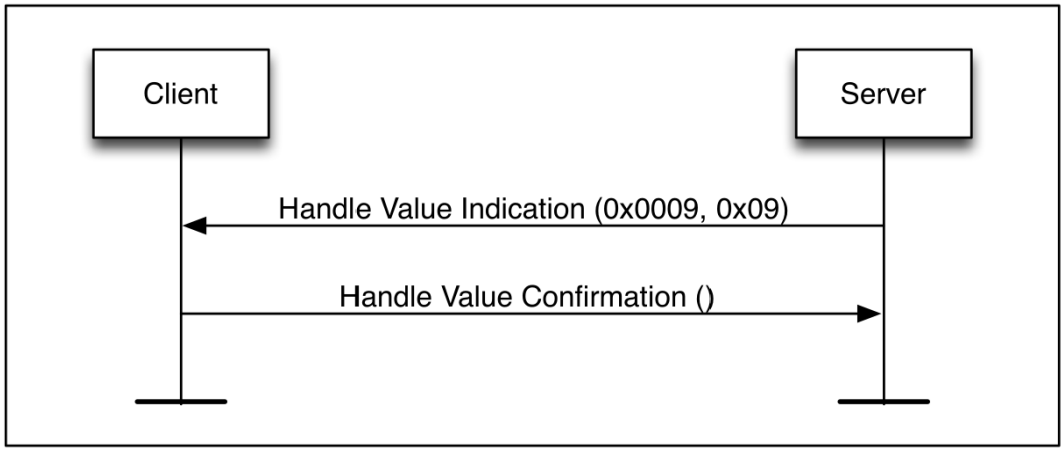



*Figure 4.18: Indications example*

## 4.12 CHARACTERISTIC DESCRIPTORS

This procedure is used to read and write characteristic descriptors on a server. There are two sub-procedures that can be used to read and write characteristic descriptors: Read Characteristic Descriptors and Write Characteristic Descriptors.

### 4.12.1 Read Characteristic Descriptors

This sub-procedure is used to read a characteristic descriptor from a server when the client knows the characteristic descriptor declaration's Attribute handle.

The Attribute Protocol *Read Request* is used for this sub-procedure. The *Read Request* is used with the *Attribute Handle* parameter set to the characteristic descriptor handle. The *Read Response* returns the characteristic descriptor value in the *Attribute Value* parameter.

An *Error Response* shall be sent by the server in response to the *Read Request* if insufficient authentication, insufficient authorization, insufficient encryption key size is used by the client, or if a read operation is not permitted on the *Characteristic Value*. The *Error Code* parameter is set accordingly.

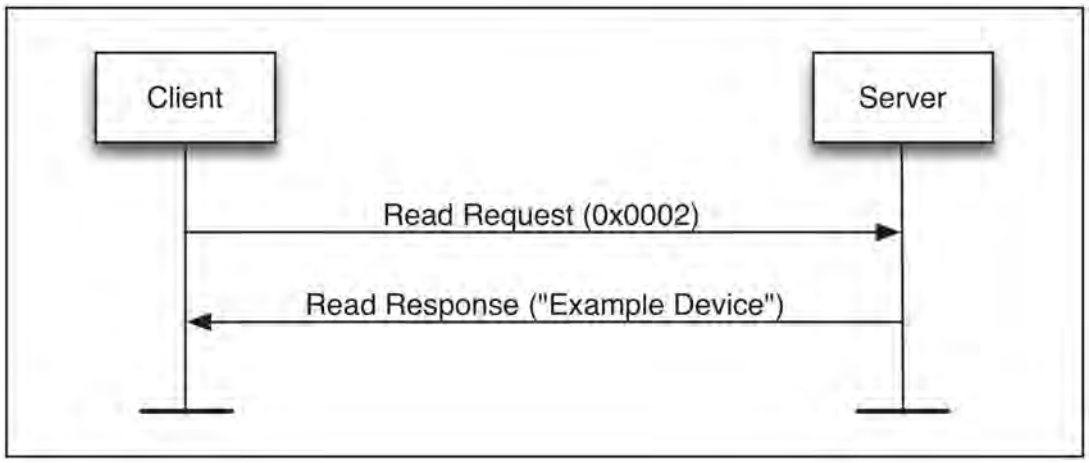



*Figure 4.19: Read Characteristic Descriptors example*

### 4.12.2 Read Long Characteristic Descriptors

This sub-procedure is used to read a characteristic descriptor from a server when the client knows the characteristic descriptor declaration's Attribute handle and the length of the characteristic descriptor declaration is longer than can be sent in a single Read Response Attribute Protocol message.

The Attribute Protocol Read Blob Request is used to perform this sub-procedure. The Attribute Handle parameter shall be set to the characteristic descriptor handle. The Value Offset parameter shall be the offset within the characteristic descriptor to be read. To read the complete characteristic descriptor the offset should be set to 0x00 for the first Read Blob Request. The offset for subsequent Read Blob Requests is the next octet that has yet to be read. The Read Blob Request is repeated until the Read Blob Response's Part Attribute Value parameter is zero or an Error Response is sent by the server with the Error Code set to Invalid Offset.

For each Read Blob Request a Read Blob Response is received with a portion of the characteristic descriptor value contained in the Part Attribute Value parameter.

An Error Response shall be sent by the server in response to the Read Blob Request if insufficient authentication, insufficient authorization, insufficient encryption key size is used by the client, or if a read operation is not permitted on the characteristic descriptor. The Error Code parameter is set accordingly.

Note: The Read Blob Request may be used to read the remainder of an characteristic descriptor value where the first part was read using a simple Read Request.

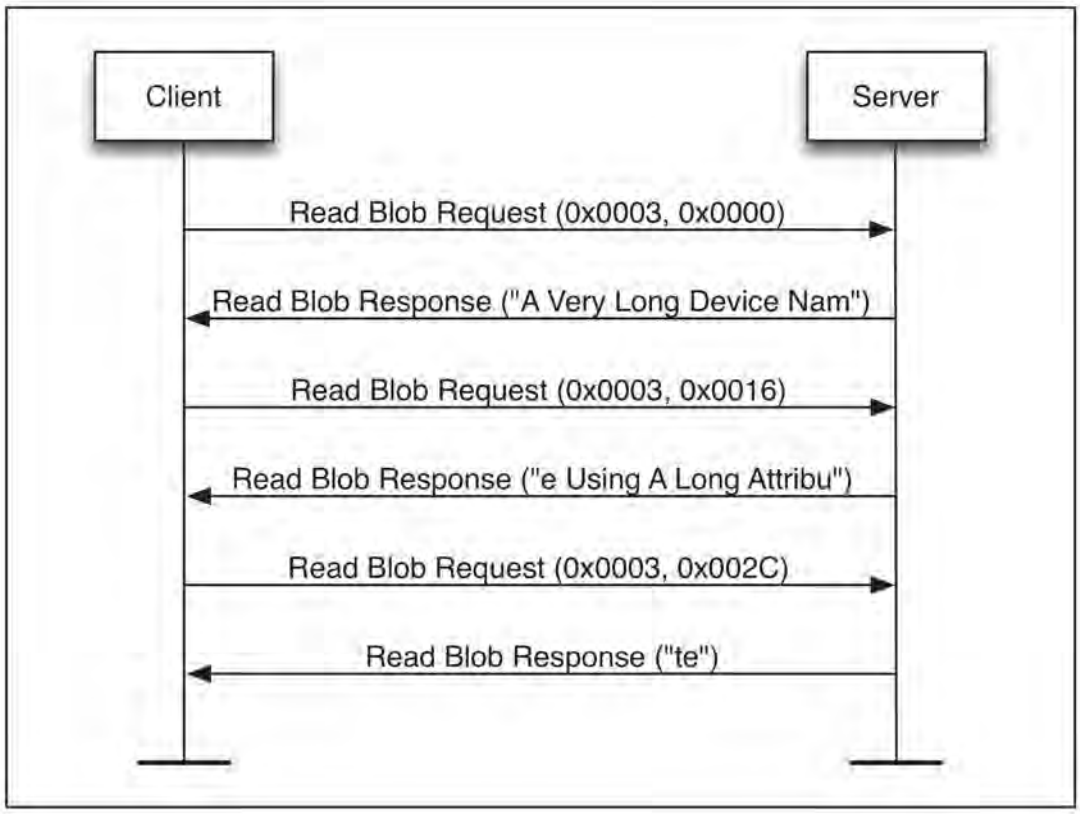



*Figure 4.20: Read Long Characteristic Descriptors example*

### 4.12.3 Write Characteristic Descriptors

This sub-procedure is used to write a characteristic descriptor value to a server when the client knows the characteristic descriptor handle.

The Attribute Protocol *Write Request* is used for this sub-procedure. The *Attribute Handle* parameter shall be set to the characteristic descriptor handle. The *Attribute Value* parameter shall be set to the new characteristic descriptor value.

A *Write Response* shall be sent by the server if the write of the characteristic descriptor value succeeded.

An *Error Response* shall be sent by the server in response to the *Write Request* if insufficient authentication, insufficient authorization, insufficient encryption key size is used by the client, or if a write operation is not permitted on the *Characteristic Value*. The *Error Code* parameter shall be set as specified in the Attribute Protocol. If the characteristic descriptor value that is written is the wrong size, or has an invalid value as defined by the profile, or the operation is not permitted at this time then the value shall not be written and an *Error Response* shall be sent with the *Error Code* set to *Application Error* by the server.

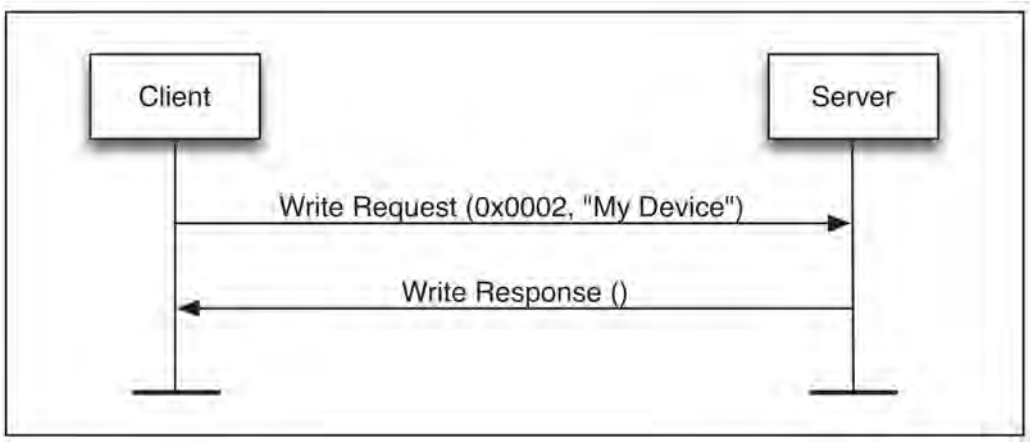



*Figure 4.21: Write Characteristic Descriptors example*

### 4.12.4 Write Long Characteristic Descriptors

This sub-procedure is used to write a characteristic descriptor value to a server when the client knows the characteristic descriptor handle but the length of the characteristic descriptor value is longer than can be sent in a single Write Request Attribute Protocol message.

The Attribute Protocol *Prepare Write Request* and *Execute Write Request* are used to perform this sub-procedure. The *Attribute Handle* parameter shall be set to the *Characteristic Descriptor Handle*. of the *Characteristic Value* to be written. The *Part Attribute Value* parameter shall be set to the part of the *Attribute Value* that is being written. The *Value Offset* parameter shall be the offset within the *Characteristic Value* to be written. To write the complete *Characteristic Value* the offset should be set to 0x0000 for the first *Prepare Write Request*. The offset for subsequent *Prepare Write Requests* is the next octet that has yet to be written. The *Prepare Write Request* is repeated until the complete *Characteristic Value* has been transferred, after which an *Executive Write Request* is used to write the complete value.

Note: The values in the Prepare Write Response do not need to be verified in this sub-procedure.

An *Error Response* shall be sent by the server in response to the *Prepare Write Request* or *Executive Write Request* if insufficient authentication, insufficient authorization, insufficient encryption key size is used by the client, or if a write operation is not permitted on the *Characteristic Value*. The *Error Code* parameter is set as specified in the Attribute Protocol. If the *Attribute Value* that is written is the wrong size, or has an invalid value as defined by the profile, then the write shall not succeed and an *Error Response* shall be sent with the *Error Code* set to *Application Error* by the server.

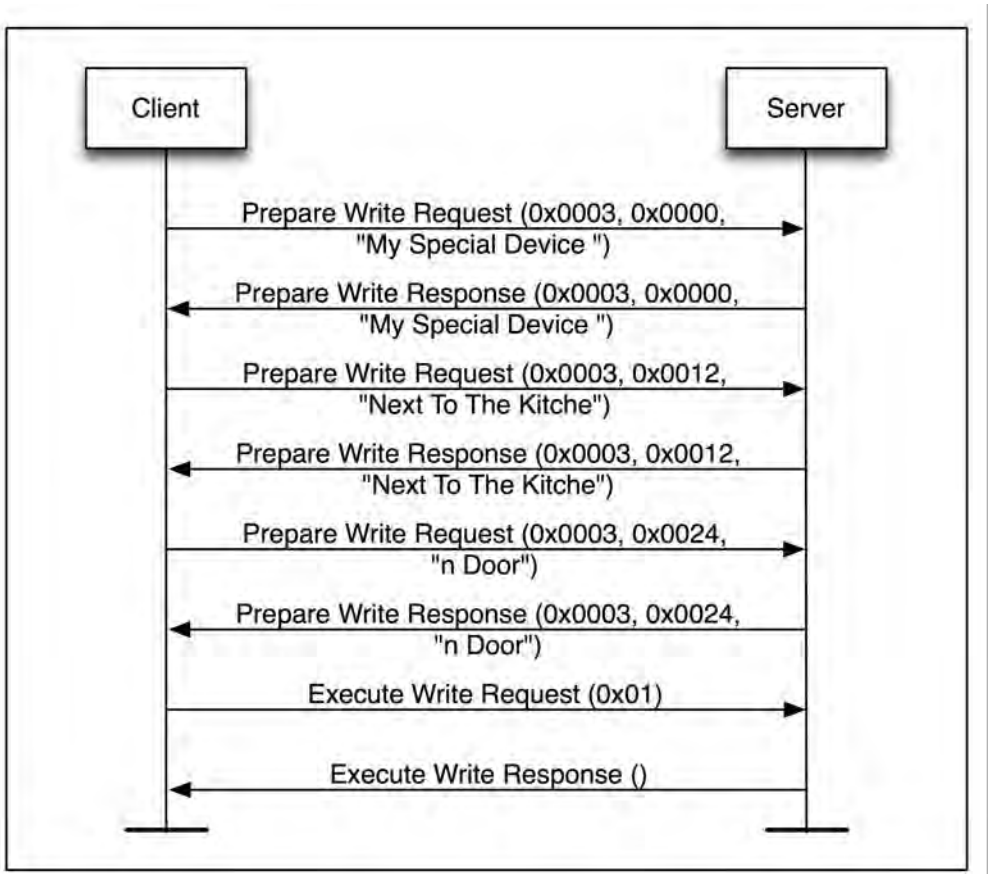



*Figure 4.22: Write Long Characteristic Descriptors example*

## 4.13 GATT PROCEDURE MAPPING TO ATT PROTOCOL OPCODES

The following table describes the mapping of the ATT protocol opcodes to the GATT procedures and sub-procedures.Only those portions of the ATT protocol requests, responses, notifications or indications necessary to implement the mandatory or supported optional sub-procedures is required.

| Feature | Sub-Procedure | ATT Protocol Opcodes |
|---|---|---|
| **Server Configuration** | Exchange MTU | Exchange MTU Request<br>Exchange MTU Response<br>Error Response |
| **Primary Service Discovery** | Discover All Primary Services | Read By Group Type Request<br>Read By Group Type Response<br>Error Response |
| | Discover Primary Services By Service UUID | Find By Type Value Request<br>Find By Type Value Response<br>Error Response |

*Table 4.2: GATT Procedure mapping to ATT protocol opcodes*

| Feature | Sub-Procedure | ATT Protocol Opcodes |
|---|---|---|
| **Relationship Discovery** | Find Included Services | Read By Type Request<br>Read By Type Response<br>Error Response |
| **Characteristic Discovery** | Discover All Characteristic of a Service | Read By Type Request<br>Read By Type Response<br>Error Response |
| | Discover Characteristic by UUID | Read By Type Request<br>Read By Type Response<br>Error Response |
| **Characteristic Descriptor Discovery** | Discover All Characteristic Descriptors | Find Information Request<br>Find Information Response<br>Error Response |
| **Characteristic Value Read** | Read Characteristic Value | Read Request<br>Read Response<br>Error Response |
| | Read Using Characteristic UUID | Read By Type Request<br>Read By Type Response<br>Error Response |
| | Read Long Characteristic Values | Read Blob Request<br>Read Blob Response<br>Error Response |
| | Read Multiple Characteristic Values | Read Multiple Request<br>Read Multiple Response<br>Error Response |

*Table 4.2: GATT Procedure mapping to ATT protocol opcodes*

| Feature | Sub-Procedure | ATT Protocol Opcodes |
|---|---|---|
| **Characteristic Value Write** | Write Without Response | Write Command |
| | Signed Write Without Response | Write Command |
| | Write Characteristic Value | Write Request<br>Write Response<br>Error Response |
| | Write Long Characteristic Values | Prepare Write Request<br>Prepare Write Response<br>Execute Write Request<br>Execute Write Response<br>Error Response |
| | Characteristic Value Reliable Writes | Prepare Write Request<br>Prepare Write Response<br>Execute Write Request<br>Execute Write Response<br>Error Response |
| **Characteristic Value Notification** | Notifications | Handle Value Notification |
| **Characteristic Value Indication** | Indications | Handle Value Indication<br>Handle Value Confirmation |
| **Characteristic Descriptor Value Read** | Read Characteristic Descriptors | Read Request<br>Read Response<br>Error Response |
| | Read Long Characteristic Descriptors | Read Blob Request<br>Read Blob Response<br>Error Response |
| **Characteristic Descriptor Value Write** | Write Characteristic Descriptors | Write Request<br>Write Response<br>Error Response |
| | Write Long Characteristic Descriptors | Prepare Write Request<br>Prepare Write Response<br>Prepare Write Request<br>Prepare Write Response<br>Error Response |

*Table 4.2: GATT Procedure mapping to ATT protocol opcodes*

## 4.14 PROCEDURE TIMEOUTS

GATT procedures are protected from failure with an Attribute Protocol transaction timeout.

If the Attribute Protocol transaction times out, the procedure shall be considered to have failed, and the local higher layer shall be notified. No further GATT procedures shall be performed. A new GATT procedure shall only be performed when a new ATT Bearer has been established.

# 5 L2CAP INTEROPERABILITY REQUIREMENTS

The following default values shall be used by an implementation of this profile. The default values used may be different depending on the physical channel that the Attribute Protocol is being sent over.

## 5.1 BR/EDR L2CAP INTEROPERABILITY REQUIREMENTS

Over BR/EDR, L2CAP connection oriented channels are used to transmit Attribute Protocol PDUs. These channels use the channel establishment procedure from L2CAP using the fixed PSM (see [1]) including the configuration procedure to determine the ATT_MTU. Therefore, the ATT Bearer (or the logical link as referred to in the Attribute Protocol) is, in this case, an established L2CAP connection-oriented channel.

### 5.1.1 ATT_MTU

At the end of the L2CAP configuration phase, upon transition to the OPEN state, the ATT_MTU for this ATT Bearer shall be set to the minimum of the negotiated Maximum Transmission Unit configuration options.

Note: the minimum ATT_MTU for BR/EDR is 48 octets, as defined by L2CAP in [Vol. 3] Part A, Section 5.1.

### 5.1.2 BR/EDR Channel Requirements

All Attribute Protocol messages sent by GATT over an L2CAP channel are sent using a dynamic channel ID derived by connecting using a fixed PSM. The use of a fixed PSM allows rapid reconnection of the L2CAP channel for Attribute Protocol as a preliminary SDP query is not required.

All packets sent on this L2CAP channel shall be Attribute PDUs.

PDUs shall be reliably sent.

The flow specification for the Attribute Protocol shall be best effort.

If operating in Basic L2CAP mode, the information payload of the L2CAP B-frame shall be a single Attribute PDU.

The channel shall be encrypted. The Key_Type shall be either an Unauthenticated Combination Key or an Authenticated Combination Key.

### 5.1.3 BR/EDR Channel Establishment Collisions

The L2CAP connection can be initiated by the client or by the server.

Only one BR/EDR L2CAP connection shall be established between a client and a server. If the L2CAP connection already exists, the client or server shall not initiate the connection request.

If both devices open an L2CAP connection simultaneously both channels shall be closed and each device shall wait a random time (not more than 1 second and not less than 100 ms) and then try to open the L2CAP connection again. If it is known which device is the master, that device can re-try at once.

## 5.2 LE L2CAP INTEROPERABILITY REQUIREMENTS

Over LE, the ATT Bearer (or logical link as referred to in the Attribute Protocol) is the Attribute L2CAP fixed channel.

Note: To remove the ATT Bearer, the physical channel has to be disconnected.

### 5.2.1 ATT_MTU

Both a GATT client and server implementations shall support an ATT_MTU not less than the default value.

| Default Value | Value for LE |
|---|---|
| ATT_MTU | 23 |

*Table 5.1: LE L2CAP ATT_MTU*

### 5.2.2 LE Channel Requirements

All Attribute Protocol messages sent by GATT over an L2CAP logical link are sent using a fixed channel ID. This enables very fast transmission of the Attribute Protocol messages once a host connection setup is complete. The use of a fixed channel ID also enables the default channel properties to be defined.

L2CAP fixed CID 0x0004 shall be used for the Attribute Protocol. All packets sent on this fixed channel shall be Attribute Protocol PDUs.

The flow specification for the Attribute Protocol shall be best effort.

PDUs shall be reliably sent, and not flushed.

The retransmission and flow control mode for this channel shall be Basic L2CAP mode

The payload of the L2CAP B-frame shall be a single Attribute PDU.

| Parameter | Value |
|---|---|
| MTU | 23 |

*Table 5.2: Attribute Protocol Fixed channel configuration parameters*

| Parameter | Value |
|---|---|
| Flush Timeout | 0xFFFF (Infinite) |
| QoS | Best Effort |
| Mode | Basic Mode |

*Table 5.2: Attribute Protocol Fixed channel configuration parameters*

# 6 GAP INTEROPERABILITY REQUIREMENTS

## 6.1 BR/EDR GAP INTEROPERABILITY REQUIREMENTS

### 6.1.1 Connection Establishment

To establish an ATT Bearer, the Channel Establishment procedure (as defined in Generic Access Profile, Section 7.2) shall be used.

Either device may terminate a link at any time.

No idle mode procedures or modes are defined by this profile.

## 6.2 LE GAP INTEROPERABILITY REQUIREMENTS

### 6.2.1 Connection Establishment

To establish a link, the Connection Establishment procedure (as defined in Generic Access Profile, Section 9.3.5 through Section 9.3.8) shall be used.

Either device may terminate a link at any time.

No idle mode procedures or modes are defined by this profile.

### 6.2.2 Profile Roles

This profile can be used in the following profile roles (as defined in Generic Access Profile):

- Central
- Peripheral

## 6.3 DISCONNECTED EVENTS

### 6.3.1 Notifications and Indications While Disconnected

If a client has configured the server to send a notification or indication to the client, it shall be configured to allow re-establishment of the connection when it is disconnected.

If the client is disconnected, but intends to become a Central in the connection it shall perform a GAP connection establishment procedure. If the client is disconnected, but intends to become a Peripheral in the connection it shall go into a GAP connectable mode.

A server shall re-establish a connection with a client when an event or trigger operation causes a notification or indication to a client.

If the server is disconnected, but intends to become a Peripheral in the connection it shall go into a GAP connectable mode. If the server is disconnected, but intends to become a Central in the connection it shall perform a GAP connection establishment procedure.

If the server cannot re-establish a connection, then the notification or indication for this event shall be discarded and no further connection re-establishment shall occur, until another event occurs.

# 7 DEFINED GENERIC ATTRIBUTE PROFILE SERVICE

All characteristics defined within this section shall be contained in a primary service with the service UUID set to «Generic Attribute Profile» as defined in Section 3.1. Only one instance of the GATT service shall be exposed on a GATT server.

Table 7.1 lists characteristics that may be present in the server and the characteristics that may be supported by the client.

| Item No. | Characteristic | Ref. | Support in Client | Support in Server |
|---|---|---|---|---|
| **1** | Service Changed | 7.1 | M | C1 |

*Table 7.1: GATT Profile service characteristic support*

C1: Mandatory if service definitions on the server can be added, changed or removed; otherwise optional

The assigned UUIDs for these characteristics are defined in [1].

## 7.1 SERVICE CHANGED

The «Service Changed» characteristic is a control-point attribute (as defined in 11) that shall be used to indicate to connected devices that services have changed (i.e., added, removed or modified). The characteristic shall be used to indicate to clients that have a trusted relationship (i.e. bond) with the server when GATT based services have changed when they re-connect to the server. See Section 2.5.2.

This *Characteristic Value* shall be configured to be indicated.

| Attribute Handle | Attribute Type | Attribute Value | | | Attribute Permission |
|---|---|---|---|---|---|
| 0xNNNN | 0x2803 – UUID for «Characteristic» | Characteristic Properties = 0x26 | 0xMMMM = Handle of Characteristic Value | 0x2A05 – UUID for «Service Changed» | No Authentication, No Authorization |

*Table 7.2: Service Changed Characteristic declaration*

The *Service Changed Characteristic Value* is two 16-bit *Attribute Handles* concatenated together indicating the beginning and ending *Attribute Handles* affected by an addition, removal, or modification to a GATT-based service on the server. If a change has been made to any of the GATT service definition characteristic values other than the *Service Changed* characteristic value, the

range shall also include the beginning and ending *Attribute Handle* for the GATT service definition.

| Attribute Handle | Attribute Type | Attribute Value | | Attribute Permission |
|---|---|---|---|---|
| 0xMMMM | 0x2A05 – UUID for «Service Changed» | 0xuuuu – Start of Affected Attribute Handle Range | 0xuuuu – End of Affected Attribute Handle Range | No Authentication, No Authorization, Not Readable, Not Writable |

*Table 7.3: Service Changed Characteristic Value declaration*

There is only one instance of the *Service Changed* characteristic within the GATT service definition. A *Service Changed* characteristic value shall exist for each client with a trusted relationship.

If the list of GATT based services and the service definitions cannot change for the lifetime of the device then this characteristic shall not exist, otherwise this characteristic shall exist.

If the *Service Changed* characteristic exists on the server, the *Characteristic Value Indication* support on the server is mandatory.

The client shall support *Characteristic Value Indication* of the *Service Changed* characteristic.

The *Service Changed* characteristic *Attribute Handle* on the server shall not change if the server has a trusted relationship with any client.

# 8 SECURITY CONSIDERATIONS

## 8.1 AUTHENTICATION REQUIREMENTS

Authentication in the GATT Profile is applied to each characteristic independently. Authentication requirements are specified in this profile, related higher layer specifications or are implementation specific if not specified otherwise.

The GATT Profile procedures are used to access information that may require the client to be authenticated and have an encrypted connection before a characteristic can be read or written.

Over LE, if such a request is issued when the physical link is unauthenticated or unencrypted, the server shall send an *Error Response* with the status code set to *Insufficient Authentication*. The client wanting to read or write this characteristic can then request that the physical link be authenticated using the GAP authentication procedure, and once this has been completed, send the request again.

Over BR/EDR, if such a request is issued when the physical link is unauthenticated or unencrypted, the server can either send an Error Response with the status code set to Insufficient Authentication or perform the GAP authentication procedure and then upon completion of that procedure send the appropriate response. If the client receives this Error Response it can then request that the physical link be authenticated using the GAP authentication procedure, and once this has been completed send the request again.

The list of services and characteristics that a device supports is not considered private or confidential information, and therefore the Service and Characteristic Discovery procedures shall always be permitted. This implies that an *Insufficient Authentication* status code shall not be used in an *Error Response* for a *Read Information Request*.

Note: A characteristic may be allowed to be read by any device, but only written by an authenticated device. An implementation should take this into account, and not assume that if it can read a *Characteristic Value*, it will also be able to write the *Characteristic Value*. Similarly, if a characteristic can be written, it does not mean the characteristic can also be read. Each individual characteristic could have different security properties.

Once sufficient authentication of the client has been established to allow access to one characteristic within a service definition, a server may also allow access to other characteristics within the service definition depending on the higher level or implementation specific requirements.

A server may allow access to most characteristics within a service definition once sufficient authentication has been performed, but restrict access to other characteristics within the same service definition. This may result due to some

characteristics requiring stronger authentication requirements than currently enabled.

Once a server has authenticated a client for access to characteristics in one service definition, it may automatically allow access to characteristics in other service definitions.

## 8.2 AUTHORIZATION REQUIREMENTS

Authorization in the GATT Profile is applied to each characteristic independently. Authorization requirements may be specified in this profile, related higher layer specifications or are implementation specific if not specified otherwise.

The GATT Profile can be used to access information that may require authorization before a characteristic can be read or written.

If such a request is issued to a characteristic contained in a service definition that is not authorized, the responder shall send an *Error Response* with the status code set to *Insufficient Authorizatio*n.

Once a server has authorized a client for access to characteristics in one group or service definition, it may automatically allow access to characteristics in other service definitions.

# 9 SDP INTEROPERABILITY REQUIREMENTS

A device that supports GATT over BR/EDR shall publish the following SDP record. The GATT Start Handle shall be set to the attribute handle of the «Generic Attribute Profile» service declaration. The GATT End Handle shall be set to the attribute handle of the last attribute within the «Generic Attribute Profile» service definition group.

| Item | Definition | Type | Value | Status |
|---|---|---|---|---|
| Service Class ID List | | | | M |
| Service Class #0 | | UUID | Generic Attribute Profile | M |
| Protocol Descriptor List | | | | M |
| Protocol #0 | | UUID | L2CAP | M |
| Parameter #0 for Protocol #0 | PSM | Uint16 | PSM = ATT | M |
| Protocol #1 | | UUID | ATT | M |
| Parameter #0 for Protocol #1 | GATT Start Handle | Uint16 | | M |
| Parameter #1 for Protocol #1 | GATT End Handle | Uint16 | | M |
| BrowseGroupList[ | | | PublicBrowseGroup | M |

*Table 9.1: SDP Record for the Generic Access Profile*

# 10 REFERENCES

[1] Assigned Numbers Specification: https://www.bluetooth.org/Technical/AssignedNumbers/home.htm

# 11 APPENDIX: EXAMPLE ATTRIBUTE SERVER ATTRIBUTES

The following table shows an example Server and the Attributes contained on the server.

| Handle | Attribute Type | Attribute Value |
|---|---|---|
| 0x0001 | «Primary Service» | «GAP Service» |
| 0x0004 | «Characteristic» | {0x02, 0x0006, «Device Name»} |
| 0x0006 | «Device Name» | “Example Device” |
| 0x0010 | «Primary Service» | «GATT Service» |
| 0x0011 | «Characteristic» | {0x02, 0x0012, «Attribute Opcodes Supported»} |
| 0x0012 | «Attribute Opcodes Supported» | 0x01FF |
| 0x0100 | «Primary Service» | «Battery State Service» |
| 0x0106 | «Characteristic» | {0x02, 0x0110, «Battery State»} |
| 0x0110 | «Battery State» | 0x04 |
| 0x0200 | «Primary Service» | «Thermometer Humidity Service» |
| 0x0201 | «Include» | {0x0500, 0x0504, «Manufacturer Service»} |
| 0x0202 | «Include» | {0x0550,0x0568} |
| 0x0203 | «Characteristic» | {0x02, 0x0203, «Temperature»} |
| 0x0204 | «Temperature» | 0x028A |
| 0x0205 | «Characteristic Format» | {0x0E, 0xFE, «Celsius», 0x01, «Outside»} |
| 0x0206 | «Characteristic User Description» | “Outside Temperature” |
| 0x0210 | «Characteristic» | {0x02, 0x0212, «Relative Humidity»} |
| 0x0212 | «Relative Humidity» | 0x27 |
| 0x0213 | «Characteristic Format» | {0x04, 0x00, «Percent», «Bluetooth SIG», «Outside»} |
| 0x0214 | «Characteristic User Description» | “Outside Relative Humidity” |
| 0x0280 | «Primary Service» | «Weight Service» |
| 0x0281 | «Include» | 0x0505, 0x0509, «Manufacturer Service»} |
| 0x0282 | «Characteristic» | {0x02, 0x0283, «Weight Kg»} |
| 0x0283 | «Weight Kg» | 0x00005582 |

*Table 11.1: Example Attribute Server Attributes*

| Handle | Attribute Type | Attribute Value |
|---|---|---|
| 0x0284 | «Characteristic Format» | {0x08, 0xFD, «Kilogram», «Bluetooth SIG», «Hanging»} |
| 0x0285 | «Characteristic User Description» | “Rucksack Weight” |
| 0x0300 | «Primary Service» | «Position Service» |
| 0x0301 | «Characteristic» | {0x02, 0x0302, «Latitude Longitude»} |
| 0x0302 | «Latitude Longitude» | 0x28BEAFA40B320FCE |
| 0x0304 | «Characteristic» | {0x02, 0x0305, «Latitude Longitude Elevation»} |
| 0x0305 | «Latitude Longitude Elevation» | 0x28BEAFA40B320FCE0176 |
| 0x0400 | «Primary Service» | «Alert Service» |
| 0x0401 | «Characteristic» | {0x0E, 0x0402, «Alert Enumeration»} |
| 0x0402 | «Alert Enumeration» | 0x00 |
| 0x0500 | «Secondary Service» | «Manufacturer Service» |
| 0x0501 | «Characteristic» | {0x02, 0x0502, «Manufacturer Name»} |
| 0x0502 | «Manufacturer Name» | “ACME Temperature Sensor” |
| 0x0503 | «Characteristic» | {0x02, 0x0504, «Serial Number»} |
| 0x0504 | «Serial Number» | “237495-3282-A” |
| 0x0505 | «Secondary Service» | «Manufacturer Service» |
| 0x0506 | «Characteristic» | {0x02, 0x0507, «Manufacturer Name»} |
| 0x0550 | «Secondary Service» | «Vendor Specific Service» |
| 0x0560 | «Characteristic» | {0x02, 0x0568, «Vendor Specific Type»} |
| 0x0568 | «Vendor Specific Type» | 0x56656E646F72 |
| 0x0507 | «Manufacturer Name» | “ACME Weighing Scales” |
| 0x0508 | «Characteristic» | {0x02, 0x0509, «Serial Number»} |
| 0x0509 | «Serial Number» | “11267-2327A00239” |

*Table 11.1: Example Attribute Server Attributes*

As can be seen, the attribute server indicates support for nine services: GAP Service, GATT Service, Battery State Service, Thermometer Humidity Service, Weight Service, Position Service, Alert Service, and two Manufacturer Services.

- The server contains the following information about each of the services:
- The characteristic containing the name of the device is “Example Device”.

- The characteristic indicating the server supports all the attribute opcodes, and supports two prepared write values.
- The characteristic containing the battery state with a value of 0x04, meaning it is discharging.
- The characteristic containing the outside temperature with a value of 6.5 °C.
- The characteristic containing the outside relative humidity with a value of 39%.
- The characteristic containing the weight hanging off the device with a value of 21.89 kg.
- The characteristic containing the position of this device with the value of 68.3585444 degrees north, 18.7830222 degrees east, with an elevation of 374 meters.
- The characteristic containing the temperature sensor manufacturer with the value of ACME Temperature Sensor.
- The characteristic containing the serial number for the temperature sensor with a value of 237495-3282-A.
- The characteristic containing the weighing sensor is manufacturer with a value of ACME Weight Scales.
- The characteristic containing the serial number for the weighing sensor with a value of 11267-2327A00239.

The device is therefore on the side of the Abisko Turiststation, Norrbottens Län, Sweden, with a battery in good state, measuring a relatively warm day, with low humidity, and a heavy rucksack.

# SECURITY MANAGER SPECIFICATION

***The Security Manager (SM) defines the protocol and behavior to manage pairing, authentication and encryption between LE devices.***


Ford Ex. 1052

# CONTENTS

**1 Introduction ..... 598**
1.1 Scope ..... 598
1.2 Conventions ..... 598
1.2.1 Bit and Byte Ordering Conventions ..... 598
**2 Security Manager ..... 599**
2.1 Introduction ..... 599
2.2 Cryptographic Toolbox ..... 600
2.2.1 Security function *e* ..... 600
2.2.2 Random Address Hash function *ah* ..... 600
2.2.3 Confirm value generation function c1 ..... 601
2.2.4 Key generation function *s1* ..... 602
2.3 Pairing Methods ..... 603
2.3.1 Security Properties ..... 604
2.3.2 IO Capabilities ..... 605
2.3.3 OOB Authentication Data ..... 606
2.3.4 Encryption Key Size ..... 606
2.3.5 Pairing Algorithms ..... 606
2.3.5.1 Selecting STK Generation Method ..... 607
2.3.5.2 Just Works ..... 608
2.3.5.3 Passkey Entry ..... 608
2.3.5.4 Out of Band ..... 609
2.3.5.5 Confirmation, STK generation, and Encryption ..... 609
2.3.6 Repeated Attempts ..... 611
2.4 Security in Bluetooth low energy ..... 611
2.4.1 Definition of Keys and Values ..... 612
2.4.2 Generation of Keys ..... 612
2.4.2.1 Generation of IRK ..... 612
2.4.2.2 Generation of CSRK ..... 613
2.4.2.3 Generation of LTK, EDIV and Rand ..... 613
2.4.3 Distribution of Keys ..... 613
2.4.4 Encrypted Session Setup ..... 614
2.4.4.1 Encryption Setup using STK ..... 614
2.4.4.2 Encryption Setup using LTK ..... 615
2.4.5 Signing Algorithm ..... 616
2.4.6 Slave Security Request ..... 617
**3 Security Manager Protocol ..... 619**
3.1 Introduction ..... 619

3.2 Security Manager Channel over L2CAP ..... 619
3.3 Command Format ..... 619
3.4 SMP Timeout ..... 620
3.5 Pairing Methods ..... 621
3.5.1 Pairing Request ..... 621
3.5.2 Pairing Response ..... 623
3.5.3 Pairing Confirm ..... 624
3.5.4 Pairing Random ..... 625
3.5.5 Pairing Failed ..... 626
3.6 Security in Bluetooth low energy ..... 627
3.6.1 Key Distribution ..... 627
3.6.2 Encryption Information ..... 629
3.6.3 Master Identification ..... 630
3.6.4 Identity Information ..... 630
3.6.5 Identity Address Information ..... 631
3.6.6 Signing Information ..... 632
3.6.7 Security Request ..... 632

**4 References ..... 634**

**5 Appendices ..... 637**
5.1 Appendix A – EDIV and Rand Generation ..... 637
5.1.1 EDIV Masking ..... 637
5.1.1.1 DIV Mask generation function dm ..... 637
5.1.1.2 EDIV Generation ..... 638
5.1.1.3 DIV Recovery ..... 638
5.2 Appendix B – Key Management ..... 638
5.2.1 Database Lookup ..... 639
5.2.2 Key Hierarchy ..... 639
5.2.2.1 Diversifying function d1 ..... 640
5.2.2.2 Generating Keys from ER ..... 641
5.2.2.3 Generating Keys from IR ..... 641
5.3 Message Sequence Charts ..... 642
5.3.1 Phase 1: Pairing Feature Exchange ..... 642
5.3.1.1 Slave Security Request– Master Requests Pairing ..... 643
5.3.2 Phase 2: Authenticating and Encrypting ..... 643
5.3.2.1 Phase 2: Short Term Key Generation – Just Works ..... 644
5.3.2.2 Phase 2: Short Term Key Generation – Passkey Entry ..... 644
5.3.2.3 Phase 2: Short Term Key Generation – Out of Band ..... 645

5.3.3 Phase 3: Transport Specific Key Distribution .....646
5.3.4 Security Re-established using Previously Distributed LTK .....647
5.3.4.1 Master Initiated Security - Master Initiated Link Layer Encryption.....647
5.3.4.2 Slave Security Request- Master Initiated Link Layer Encryption.....647
5.3.5 Failure Conditions .....648
5.3.5.1 Pairing Not Supported by Slave.....648
5.3.5.2 Master Rejects Pairing Because of Key Size648
5.3.5.3 Slave Rejects Pairing Because of Key Size..649
5.3.5.4 Passkey Entry Failure on Master.....649
5.3.5.5 Passkey Entry Failure on Slave.....650
5.3.5.6 Slave Rejects Master's Confirm Value.....651
5.3.5.7 Master Rejects Slaves Confirm Value.....651

# 1 INTRODUCTION

## 1.1 SCOPE

The Security Manager defines methods of pairing and key distribution, a protocol for those methods and a security toolbox to be used by those methods and other parts of an LE device.

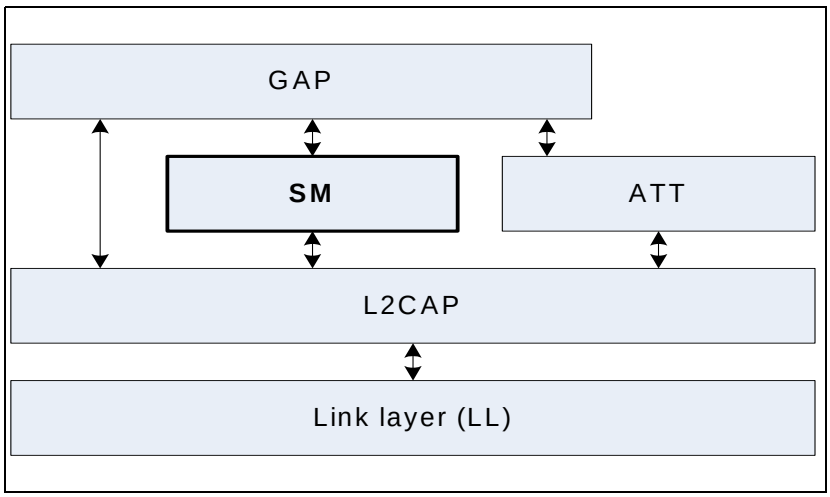



*Figure 1.1: Relationship of the Security Manager to the rest of the LE Bluetooth architecture*

The document describes master and slave roles in terms of protocol and requirements; these have the same meaning and are mapped to the LE device roles described in [Vol 1] Part A, Section 1.2 or BR/EDR device roles (see [Vol 1] Part A, Section 1.1).

## 1.2 CONVENTIONS

### 1.2.1 Bit and Byte Ordering Conventions

When multiple bit fields are contained in a single octet and represented in a drawing in this specification, the least significant (low-order) bits are shown toward the left and most significant (high-order) bits toward the right.

Multiple-octet fields are drawn with the least significant octets toward the left and the most significant octets toward the right. Multiple-octet fields shall be transmitted with the least significant octet first.

Multiple-octet values written in hexadecimal notation have the most significant octet towards the left and the least significant octet towards the right, for example if ‘12’ is the most significant octet and ‘34’ is the least significant octet it would be written as 0x1234.

# 2 SECURITY MANAGER

## 2.1 INTRODUCTION

The Security Manager (SM) uses a key distribution approach to perform identity and encryption functionalities in radio communication. This means that each device generates and controls the keys it distributes and no other device affects the generation of these keys. The strength of a key is as strong as the algorithms implemented inside the distributing device.

The security architecture is designed such that memory and processing requirements for a responding device are lower than the memory and processing requirement for an initiating device.

Pairing is performed to establish keys which can then be used to encrypt a link. A transport specific key distribution is then performed to share the keys which can be used to encrypt a link in future reconnections, verify signed data and random address resolution.

Pairing is a three-phase process. The first two phases are always used and may be followed by an optional transport specific key distribution phase (see Figure 2.1):

- Phase 1: Pairing Feature Exchange
- Phase 2: Short Term Key (STK) Generation
- Phase 3: Transport Specific Key Distribution

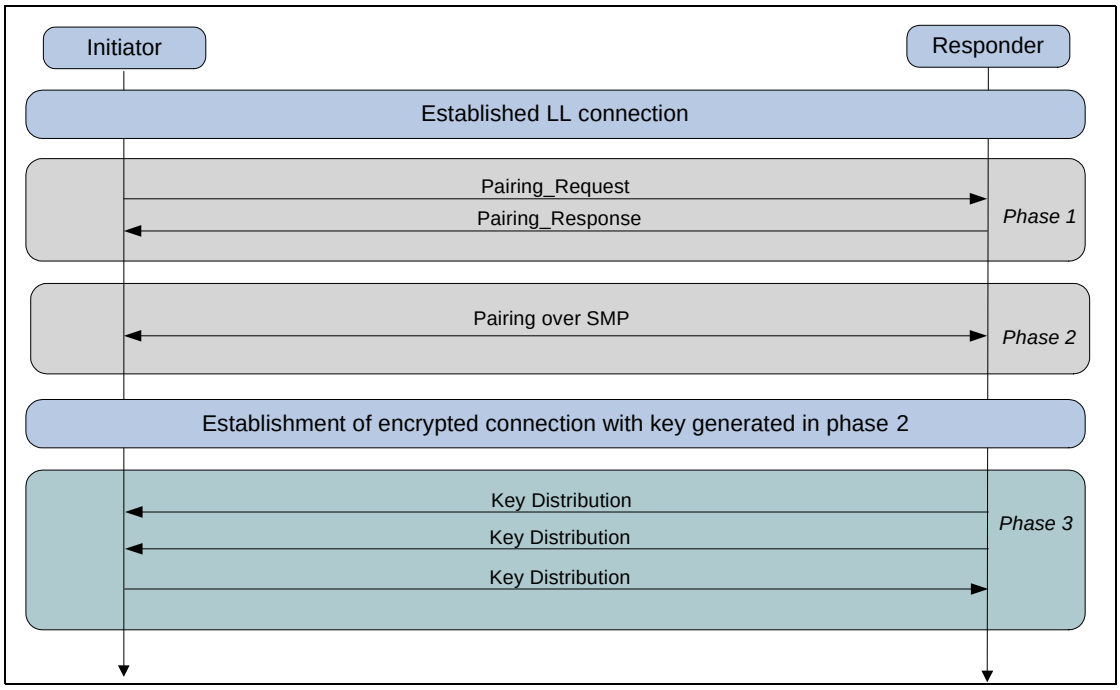



*Figure 2.1: LE Pairing Phases*

The devices shall first exchange IO capabilities in the Pairing Feature Exchange to determine which of the following methods shall be used in Phase 2:

- Just Works
- Passkey Entry
- Out Of Band (OOB)

Optionally, Phase 3 may then be performed to distribute transport specific keys, for example the Long Term Key (LTK) and Encrypted Diversifier (EDIV) values. Phase 1 and Phase 3 are identical regardless of the method used in Phase 2.

Phase 3 shall only be performed on a link which is encrypted using the STK generated in Phase 2. Phase 1 and Phase 2 may be performed on a link which is either encrypted or not encrypted.

## 2.2 CRYPTOGRAPHIC TOOLBOX

In order to support random addressing, pairing and other operations SM provides a toolbox of cryptographic functions. The following cryptographic functions are defined:

- *ah* is used to create a 24-bit hash used in random address creation and resolution.
- *c1* is used to generate confirm values used during the pairing process.
- *s1* is used to generate the STK during the pairing process.

The building block for all the above cryptographic functions is the security function *e*.

### 2.2.1 Security function *e*

Security function *e* generates 128-bit *encryptedData* from a 128-bit key and 128-bit *plaintextData* using the AES-128-bit block cypher as defined in [1]:

*encryptedData* = *e*(*key*, *plaintextData*)

The most significant octet of *key* corresponds to key[0], the most significant octet of *plaintextData* corresponds to in[0] and the most significant octet of *encryptedData* corresponds to out[0] using the notation specified in [1].

Note: The security function *e* can be implemented in a Host or be implemented using the HCI_LE_Encrypt command (see [Vol 2] Part E, Section 7.8.22).

### 2.2.2 Random Address Hash function *ah*

The random address hash function *ah* is used to generate a hash value that is used in resolvable private addresses, see Part C, Section 10.8.2.

The following are inputs to the random address hash function *ah*:

k is 128 bits

r is 24 bits

padding is 104 bits

*r* is concatenated with padding to generate *r*' which is used as the 128-bit input parameter *plaintextData* to security function *e*:

r' = padding || r

The least significant octet of *r* becomes the least significant octet of r*'* and the most significant octet of *padding* becomes the most significant octet of *r'*.

For example, if the 24-bit value *r* is 0x423456 then *r'* is 0x000000000000000000000000423456.

The output of the random address function *ah* is:

$ah(k, r) = e(k, r') \bmod 2^{24}$

The output of the security function *e* is then truncated to 24 bits by taking the least significant 24 bits of the output of *e* as the result of *ah*.

### 2.2.3 Confirm value generation function c1

During the pairing process confirm values are exchanged. This confirm value generation function *c1* is used to generate the confirm values.

The following are inputs to the confirm value generation function *c1*:

k is 128 bits

r is 128 bits

pres is 56 bits

preq is 56 bits

iat is 1 bit

ia is 48 bits

rat is 1 bit

ra is 48 bits

padding is 32 bits or 0

*iat* is concatenated with 7-bits of 0 to create *iat'* which is 8 bits in length. *iat* is the least significant bit of *iat'*.

*rat* is concatenated 7-bits of 0 to create *rat'* which is 8 bits in length. *rat* is the least significant bit of *rat'*.

*pres*, *preq*, *rat'* and *iat'* are concatenated to generate *p1* which is XORed with *r* and used as 128-bit input parameter *plaintextData* to security function *e*:

p1 = pres || preq || rat' || iat'

The octet of *iat'* becomes the least significant octet of *p1* and the most significant octet of *pres* becomes the most significant octet of *p1*.

For example, if the 8-bit *iat'* is 0x01, the 8-bit *rat'* is 0x00, the 56-bit *preq* is 0x07071000000101 and the 56 bit *pres* is 0x05000800000302 then *p1* is 0x05000800000302070710000001010001.

*ra* is concatenated with *ia* and *padding* to generate *p2* which is XORed with the result of the security function *e* using *p1* as the input parameter *plaintextData* and is then used as the 128-bit input parameter *plaintextData* to security function *e*:

p2 = padding || ia || ra

The least significant octet of *ra* becomes the least significant octet of *p2* and the most significant octet of *padding* becomes the most significant octet of *p2*.

For example, if 48-bit *ia* is 0xA1A2A3A4A5A6 and the 48-bit *ra* is 0xB1B2B3B4B5B6 then *p2* is 0x00000000A1A2A3A4A5A6B1B2B3B4B5B6.

The output of the confirm value generation function *c1* is:

c1 (k, r, preq, pres, iat, rat, ia, ra) = e(k, e(k, r XOR p1) XOR p2)

The 128-bit output of the security function *e* is used as the result of confirm value generation function *c1.*

For example, if the 128-bit k is 0x00000000000000000000000000000000, the 128-bit value r is 0x5783D52156AD6F0E6388274EC6702EE0, the 128-bit value p1 is 0x05000800000302070710000001010001 and the 128-bit value p2 is 0x00000000A1A2A3A4A5A6B1B2B3B4B5B6 then the 128-bit output from the c1 function is 0x1e1e3fef878988ead2a74dc5bef13b86.

### 2.2.4 Key generation function *s1*

The key generation function *s1* is used to generate the STK during the pairing process.

The following are inputs to the key generation function *s1* :

*k* is 128 bits

*r1* is 128 bits

*r2* is 128 bits

The most significant 64-bits of r1 are discarded to generate r1' and the most significant 64-bits of r2 are discarded to generate r2'.

For example if the 128-bit value r1 is 0x000F0E0D0C0B0A091122334455667788 then r1' is 0x1122334455667788. If the 128-bit value r2 is 0x0102030405060708 99AABBCCDDEEFF00 then r2' is 0x99AABBCCDDEEFF00.

*r1'* is concatenated with *r2'* to generate *r*' which is used as the 128-bit input parameter *plaintextData* to security function *e*:

*r*' = *r1'* || *r2*'

The least significant octet of *r2'* becomes the least significant octet of r*'* and the most significant octet of *r1'* becomes the most significant octet of *r'*.

For example, if the 64-bit value *r1'* is 0x1122334455667788 and *r2'* is 0x99AABBCCDDEEFF00 then *r'* is 0x11223344556677 8899AABBCCDDEEFF00.

The output of the key generation function *s1* is:

*s1*(k, *r1*, *r2*) = e(k, r')

The 128-bit output of the security function e is used as the result of key generation function *s1*.

For example if the 128-bit value k is

0x00000000000000000000000000000000

and the 128-bit value r' is

0x11223344556677 8899AABBCCDDEEFF00

then the output from the key generation function *s1* is

0x9a1fe1f0e8b0f49b5b4216ae796da062.

## 2.3 PAIRING METHODS

When pairing is started, the Pairing Feature Exchange shall be initiated by the initiating device. If the responding device does not support pairing or pairing cannot be performed then the responding device shall respond using the Pairing Failed message with the error code "Pairing Not Supported" otherwise it responds with a Pairing Response message.

The Pairing Feature Exchange is used to exchange IO capabilities, OOB authentication data availability, authentication requirements, key size requirements and which transport specific keys to distribute. The IO capabilities, OOB authentication data availability and authentication requirements are used to determine the STK Generation method used in Phase 2.

All the pairing methods use and generate 2 keys:

1. Temporary Key (TK): a 128-bit temporary key used in the pairing process which is used to generate STK (see Section 2.3.5.5).

2. Short Term Key (STK): a 128-bit temporary key used to encrypt a connection following pairing.

Authentication requirements are set by GAP, (see [Part C], Generic Access Profile, Section 10.3.The authentication requirements include the type of bonding and man-in-the-middle protection (MITM) requirements.

The initiating device indicates to the responding device which transport specific keys it would like to send to the responding device and which keys it would like the responding device to send to the initiator. The responding device replies with the keys that the initiating device shall send and the keys that the responding device shall send. The keys that can be distributed are defined in Section 2.4.3. If the device receives a command with invalid parameters, it shall respond with Pairing Failed command with the error code “Invalid Parameters.”

### 2.3.1 Security Properties

Security properties provided by SM are classified into one of the following categories:

- Authenticated MITM protection
- Unauthenticated no MITM protection
- No security requirements

Authenticated man-in-the-middle (MITM) protection is obtained by using the passkey entry pairing method or may be obtained using the out of band pairing method. To ensure that Authenticated MITM Protection is generated, the selected Authentication Requirements option must have MITM protection specified.

Unauthenticated no MITM Protection does not have protection against MITM attacks.

None of the pairing methods provide protection against a passive eavesdropper during the pairing process as predictable or easily established values for *TK* are used. If the pairing information is distributed without an eavesdropper being present then all the pairing methods provide confidentiality.

Note: A future version of this specification will include elliptic curve cryptography and Diffie-Hellman public key exchanges that will provide passive eavesdropper protection.

An initiating device shall maintain a record of the Security Properties for the distributed keys in a security database.

A responding device may maintain a record of the distributed key sizes and Security Properties for the distributed keys in a security database. Depending upon the key generation method and negotiated key size a responding device

may have to reduce the key length (see Section 2.3.4) so that the initiator and responder are using identical keys.

### 2.3.2 IO Capabilities

Input and output capabilities of a device are combined to generate its IO capabilities. The input capabilities are described in Table 2.1. The output capabilities are described in Table 2.2.

| Capability | Description |
|---|---|
| No input | Device does not have the ability to indicate ‘yes’ or ‘no’ |
| Yes / No | Device has at least two buttons that can be easily mapped to 'yes' and 'no' or the device has a mechanism whereby the user can indicate either 'yes' or 'no' (see note below). |
| Keyboard | Device has a numeric keyboard that can input the numbers '0' through '9' and a confirmation. Device also has at least two buttons that can be easily mapped to 'yes' and 'no' or the device has a mechanism whereby the user can indicate either 'yes' or 'no' (see note below). |

*Table 2.1: User Input Capabilities*

Note: 'yes' could be indicated by pressing a button within a certain time limit otherwise 'no' would be assumed.

| Capability | Description |
|---|---|
| No output | Device does not have the ability to display or communicate a 6 digit decimal number |
| Numeric output | Device has the ability to display or communicate a 6 digit decimal number |

*Table 2.2: User Output Capabilities*

The individual input and output capabilities are mapped to a single IO capability for that device which is used in the pairing feature exchange. The mapping is described in Table 2.3.

| Local output capacity / Local input capacity | No output | Numeric output |
|---|---|---|
| No input | NoInputNoOutput | DisplayOnly |
| Yes/No | NoInputNoOutput[1] | DisplayYesNo |
| Keyboard | KeyboardOnly | KeyboardDisplay |

*Table 2.3: I/O Capabilities Mapping*

1. None of the pairing algorithms can use Yes/No input and no output, therefore NoInputNoOutput is used as the resulting IO capability.

### 2.3.3 OOB Authentication Data

An out of band mechanism may be used to communicate information which is used during the pairing process.

If both of the devices have out of band authentication data, then this shall be used to authenticate the devices. The OOB data flag shall be set if a device has out of band authentication data.

### 2.3.4 Encryption Key Size

Each device shall have maximum and minimum encryption key length parameters which defines the maximum and minimum size of the encryption key allowed in octets. The maximum and minimum encryption key length parameters shall be between 7 octets (56 bits) and 16 octets (128 bits), in 1 octet (8 bit) steps. This is defined by a profile or device application.

The smaller value of the initiating and responding devices maximum encryption key length parameters shall be used as the encryption key size.

Both the initiating and responding devices shall check that the resultant encryption key size is not smaller than the minimum key size parameter for that device and if it is, the device shall send the Pairing Failed command with error code “Encryption Key Size.”.

The encryption key size may be stored so it can be checked by any service that has minimum encryption key length requirements.

If a key has an encryption key size that is less than 16 octets (128 bits), it shall be created by masking the appropriate MSBs of the generated key to provide a resulting key that has the agreed encryption key size. The masking shall be done after generation and before being distributed (in the case of LTK), used or stored.

For example, if a 128-bit encryption key is

0x123456789ABCDEF0123456789ABCDEF0

and it is reduced to 7 octets (56 bits), then the resulting key is

0x00000000000000000003456789ABCDEF0.

### 2.3.5 Pairing Algorithms

The information exchanged in Phase 1 is used to select which STK generation method is used in Phase 2.

The pairing is performed by each device generating a Temporary Key (*TK*). The method to generate *TK* depends upon the pairing method chosen using the algorithm described in Section 2.3.5.1. If Just Works is used then *TK* shall be generated as defined in Section 2.3.5.2. If Passkey Entry is used then *TK* shall be generated as defined in Section 2.3.5.3. If Out Of Band is used then *TK* shall be generated as defined in Section 2.3.5.4.

The *TK* value shall be used in the authentication mechanism defined in Section 2.3.5.5 to generate the STK and encrypt the link.

#### *2.3.5.1 Selecting STK Generation Method*

If both devices have out of band authentication data, then the Authentication Requirements Flags shall be ignored when selecting the pairing method and the Out of Band pairing method shall be used. If both devices have not set the MITM option in the Authentication Requirements Flags, then the IO capabilities shall be ignored and the Just Works association model shall be used. Otherwise the IO capabilities of the devices shall be used to determine the pairing method as defined in Table 2.4.

| Initiator / Responder | DisplayOnly | Display YesNo | Keyboard Only | NoInputNo Output | Keyboard Display |
|---|---|---|---|---|---|
| DisplayOnly | Just Works<br>Unauthenticated | Just Works<br>Unauthenticated | Passkey Entry: responder displays, initiator inputs<br>Authenticated | Just Works<br>Unauthenticated | Passkey Entry: responder displays, initiator inputs<br>Authenticated |

*Table 2.4: Mapping of IO Capabilities to STK Generation Method*

| Initiator / Responder | DisplayOnly | Display YesNo | Keyboard Only | NoInputNo Output | Keyboard Display |
|---|---|---|---|---|---|
| Display YesNo | Just Works<br>Unauthenticated | Just Works<br>Unauthenticated | Passkey Entry: responder displays, initiator inputs<br>Authenticated | Just Works<br>Unauthenticated | Passkey Entry: responder displays, initiator inputs<br>Authenticated |
| Keyboard-Only | Passkey Entry: initiator displays, responder inputs<br>Authenticated | Passkey Entry: initiator displays, responder inputs<br>Authenticated | Passkey Entry: initiator and responder inputs<br>Authenticated | Just Works<br>Unauthenticated | Passkey Entry: initiator displays, responder inputs<br>Authenticated |
| NoInputNo-Output | Just Works<br>Unauthenticated | Just Works<br>Unauthenticated | Just Works<br>Unauthenticated | Just Works<br>Unauthenticated | Just Works<br>Unauthenticated |
| Keyboard-Display | Passkey Entry: initiator displays, responder inputs<br>Authenticated | Passkey Entry: initiator displays, responder inputs<br>Authenticated | Passkey Entry: responder displays, initiator inputs<br>Authenticated | Just Works<br>Unauthenticated | Passkey Entry: initiator displays, responder inputs<br>Authenticated |

*Table 2.4: Mapping of IO Capabilities to STK Generation Method*

The generated key will either be an Authenticated MITM Protection or Unauthenticated no MITM Protection key. If the out of band authentication method is used the key is assumed to be Authenticated MITM Protection; however, the exact strength depends upon the method used to transfer the out of band information. The mapping of IO capabilities to an authenticated or unauthenticated key is described in Table 2.4.

If the initiating device has Out of Band data and the responding device does not have Out of Band data then the responding device may send the Pairing Failed command with the error code “OOB Not Available” instead of the Pairing Response command.

If the STK generation method does not result in an STK that provides sufficient security properties then the device shall send the Pairing Failed command with the error code “Authentication Requirements”.

### *2.3.5.2 Just Works*

The Just Works STK generation method provides no protection against eavesdroppers or man in the middle attacks during the pairing process. If the attack is not present during the pairing process then confidentiality can be established by using encryption on a future connection.

Both devices set the TK value used in the authentication mechanism defined in Section 2.3.5.5 to zero.

### *2.3.5.3 Passkey Entry*

The Passkey Entry STK generation method uses 6 numeric digits passed out of band by the user between the devices. A 6 digit numeric randomly generated passkey achieves approximately 20 bits of entropy.

If the IO capabilities of a device are DisplayOnly or if Table 2.4 defines that the device displays the passkey, then that device shall display a randomly generated passkey value between 000,000 and 999,999. The display shall ensure that all 6 digits are displayed – including zeros. The other device shall allow the user to input a value between 000,000 and 999,999.

If entry of Passkey in UI fails to occur or is canceled then the device shall send Pairing Failed command with reason code “Passkey Entry Failed”.

For example, if the user entered passkey is ‘019655’ then *TK* shall be 0x00000000000000000000000000004CC7.

The passkey Entry method provides protection against active “man-in-the-middle” (MITM) attacks as an active man-in-the-middle will succeed with a probability of 0.000001 on each invocation of the method.

The Passkey Entry STK generation method provides very limited protection against eavesdroppers during the pairing process because of the limited range of possible TK values which STK is dependent upon. If the attacker is not present during the pairing process then confidentiality and authentication can be established by using encryption on a future connection.

The TK value shall then be used in the authentication mechanism defined in Section 2.3.5.5.

### *2.3.5.4 Out of Band*

An out of band mechanism may be used to communicate information to help with device discovery, for example device address, and the 128-bit *TK* value used in the pairing process. The *TK* value shall be a 128-bit random number using the requirements for random generation defined in [Vol 2] Part H, Section 2.

If the OOB communication is resistant to MITM attacks, then this association method is also resistant to MITM attacks. Also, in the Out of Band method, the size of authentication parameter (*TK*) need not be restricted by what the user can comfortably read or type. For that reason, the Out of Band method can be more secure than using the Passkey Entry or Just Works methods. However, both devices need to have matching OOB interfaces.

MITM protection is only provided if an active man-in-the-middle chance of a successful attack has a probability of 0.000001 or less in succeeding.

#### *2.3.5.5 Confirmation, STK generation, and Encryption*

The initiating device generates a 128-bit random number (*Mrand*).

The initiating device calculates the 128-bit confirm value (*Mconfirm*) using the confirm value generation function *c1* (see Section 2.2.3) with the input parameter *k* set to *TK*, the input parameter *r* set to *Mrand,* the input parameter *preq* set to Pairing Request command, the input parameter *pres* set to the Pairing Response command, the input parameter *iat* set to the initiating device address type, *ia* set to the initiating device address, *rat* set to the responding device address type and *ra* set to the responding device address:

> *Mconfirm* = *c1*(*TK*, *Mrand*,
> Pairing Request command, Pairing Response command,
> initiating device address type, initiating device address,
> responding device address type, responding devices address)

Initiating and responding device addresses used for confirmation generation shall be device addresses used during connection setup, see [Part C], Generic Access Profile, Section 9.3

The responding device generates a 128-bit random number (*Srand*).

The responding device calculates the 128-bit confirm value (Sconfirm) using the confirm value generation function c1 (see Section 2.2.3) with the input parameter *k* set to *TK*, the input parameter *r* set to *Srand*,the input parameter *preq* set to Pairing Request command, the input parameter *pres* set to the Pairing Response command, the input parameter *iat* set to the initiating device address type, *ia* set to the initiating device address, *rat* set to the responding device address type and *ra* set to the initiating devices address:

> *Sconfirm* = *c1*(*TK*, *Srand*,
> Pairing Request command, Pairing Response command,
> initiating device address type, initiating devices address,
> responding device address type, responding device address)

The initiating device transmits *Mconfirm* to the responding device. When the responding device receives *Mconfirm* it transmits *Sconfirm* to the initiating device. When the initiating device receives *Sconfirm* it transmits *Mrand* to the responding device.

The responding device verifies the *Mconfirm* value by repeating the calculation the initiating device performed, using the *Mrand* value received.

If the responding device's calculated *Mconfirm* value does not match the received *Mconfirm* value from the initiating device then the pairing process shall be aborted and the responding device shall send the Pairing Failed command with reason code "Confirm Value Failed".

If the responding device's calculated *Mconfirm* value matches the received *Mconfirm* value from the initiating device the responding device transmits *Srand* to the initiating device.

The initiating device verifies the received *Sconfirm* value by repeating the calculation the responding device performed, using the *Srand* value received.

If the initiating devices calculated *Sconfirm* value does not match the received *Sconfirm* value from the responding device then the pairing process shall be aborted and the initiating device shall send the Pairing Failed command with the reason code "Confirm Value Failed".

If the initiating device's calculated *Sconfirm* value matches the received *Sconfirm* value from the responding device the initiating device then calculates STK and tells the Controller to enable encryption.

STK is generated using the key generation function *s1* defined in Section 2.2.4 with the input parameter *k* set to *TK*, the input parameter r1 set to *Srand,* and the input parameter *r2* set to *Mrand*:

STK = *s1*(*TK*, *Srand*, *Mrand*)

If the encryption key size is less than 128 bits then the STK shall be masked to the correct key size as described in Section 2.3.4.

The initiator shall use the generated STK to either enable encryption on the link or if encryption has already been enabled, perform the encryption pause procedure (see Section 2.4.4.1).

### 2.3.6 Repeated Attempts

When a pairing procedure fails a waiting interval shall pass before the verifier will initiate a new Pairing Request command or Security Request command to the same claimant, or before it will respond to a Pairing Request command or Security Request command initiated by a device claiming the same identity as the failed device. For each subsequent failure, the waiting interval shall be increased exponentially. That is, after each failure, the waiting interval before a new attempt can be made, could be for example, twice as long as the waiting

interval prior to the previous attempt[1]. The waiting interval should be limited to a maximum.

The maximum waiting interval depends on the implementation. The waiting time shall exponentially decrease to a minimum when no new failed attempts are made during a certain time period. This procedure prevents an intruder from repeating the pairing procedure with a large number of different keys.

## 2.4 SECURITY IN BLUETOOTH LOW ENERGY

Security shall be initiated by the Security Manager in the device in the master role. The device in the slave role shall be the responding device. The slave device may request the master device to initiate pairing or other security procedures, see Section 2.4.6.

The slave in the key distribution phase gives keys to the master so a reconnection can be encrypted, its random addresses can be resolved, or the master device can verify signed data from the slave.

The master may also provide keys to the slave device so a reconnection can be encrypted if the roles are reversed, the master's random addresses can be resolved, or the slave can verify signed data from the master.

### 2.4.1 Definition of Keys and Values

LE security uses the following keys and values for encryption, signing, and random addressing:

1. Identity Resolving Key (IRK) is a 128-bit key used to generate and resolve random addresses.
2. Connection Signature Resolving Key (CSRK) is a 128-bit key used to sign data and verify signatures on the receiving device.
3. Long Term Key (LTK) is a 128-bit key used to generate the contributory session key for an encrypted connection. Link Layer encryption is described in [Vol 6] Part B, Section 5.1.3.
4. Encrypted Diversifier (EDIV) is a 16-bit stored value used to identify the LTK. A new EDIV is generated each time a unique LTK is distributed.
5. Random Number (Rand) is a 64-bit stored valued used to identify the LTK. A new Rand is generated each time a unique LTK is distributed.

### 2.4.2 Generation of Keys

The keys used in SM are distributed rather than contributed to by both devices.

1. Another appropriate integer value larger than 1 may be used.

Any method of generation of LTK, CSRK, or IRK that results in the keys having 128 bits of entropy can be used, as the generation method is not visible outside the slave device (see Section 5.2.)The keys shall not be generated only from information that is distributed to the master device or only from information that is visible outside of the slave device.

#### *2.4.2.1 Generation of IRK*

The Identity Resolving Key (IRK) is used for resolvable private address construction (see [Part C], Generic Access Profile, Section 10.8.2. A master that has received IRK from a slave can resolve that slave's random device addresses. A slave that has received IRK from a master can resolve that master's random device addresses. The privacy concept only protects against devices that are not part of the set to which the IRK has been given.

IRK can be assigned, or randomly generated by the device during manufacturing, or some other method could be used. If IRK is randomly generated then the requirements for random generation defined in [Vol 2] Part H, Section 2 shall be used.

The encryption key size does not apply to IRK; therefore, its size does not need to be reduced before distribution.

#### *2.4.2.2 Generation of CSRK*

The Connection Signature Resolving Key (CSRK) is used to sign data in a connection. A device that has received CSRK can verify signatures generated by the distributing device. The signature only protects against devices that are not part of the set to which CSRK has been given.

CSRK can be assigned or randomly generated by the device during manufacturing, or some other method could be used. If CSRK is randomly generated then the requirements for random generation defined in [Vol 2] Part H, Section 2 shall be used.

The encryption key size does not apply to CSRK, therefore its size does not need to be reduced before distribution.

#### *2.4.2.3 Generation of LTK, EDIV and Rand*

Devices which support encryption in the Link Layer Connection State in the Slave Role shall be capable of generating LTK, EDIV, and Rand.

The EDIV and Rand are used by the slave device to establish a previously shared LTK in order to start an encrypted connection with a previously paired master device.

The generated LTK size must not exceed the negotiated encryption key size and its size may need to be reduced (see Section 2.3.4).

New values of LTK, EDIV, and Rand shall be generated each time they are distributed.

The slave device shall store the mapping between EDIV, Rand and LTK in a security database so the correct LTK value is used when the master device requests encryption. Depending upon the LTK generation method additional information may need to be stored, for example the size of the distributed LTK.

The master device may also distribute EDIV, Rand, and LTK to the slave device which can be used to encrypt a reconnection if the device roles are reversed in a future connection.

### 2.4.3 Distribution of Keys

The slave may distribute to the master the following keys:

- LTK, EDIV, and Rand
- IRK
- CSRK

The master device may distribute to the slave the following keys:

- LTK, EDIV, and Rand
- IRK
- CSRK

The security properties of the distributed keys shall be set to the security properties of the STK that was used to distribute them. For example if STK has Unauthenticated no MITM Protection security properties then the distributed keys shall have Unauthenticated no MITM Protection security properties.

The link shall be encrypted or re-encrypted using STK generated in Phase 2 (see Section 2.4.4.1) before any keys are distributed.

Note: The distributed EDIV and Rand values are transmitted in clear text by the master device to the slave device during encrypted session setup.

The BD_ADDR that is received in the Identity Address Information command shall only be considered valid once a reconnection has occurred using the BD_ADDR and LTK distributed during that pairing. Once this is successful the BD_ADDR and the distributed keys shall be associated with that device in the security database.

A device may request encrypted session setup to use the LTK, EDIV, and Rand values distributed by the slave device when the key distribution phase has completed; however, this does not provide any additional security benefit.

If an attacker has established the distributed LTK value then performing encrypted session setup to use the distributed values does not provide any protection against that attacker.

### 2.4.4 Encrypted Session Setup

During the encryption session setup the master device sends a 16-bit Encrypted Diversifier value, *EDIV,* and a 64-bit Random Number, *Rand*, distributed by the slave device during pairing, to the slave device. The master's Host provides the Link Layer with the Long Term Key to use when setting up the encrypted session. The slave's Host receives the *EDIV* and *Rand* values and provides a Long Term Key to the slaves Link Layer to use when setting up the encrypted link.

#### *2.4.4.1 Encryption Setup using STK*

To distribute LTK and other keys in pairing Phase 3 an encrypted session needs to be established (see Section 2.3.5.5).

The encrypted session is setup using STK generated in Phase 2 (see Section Section 2.3.5.5) as the Long Term Key provided to the Link Layer, (see [Vol 6] Part B, Section 5.1.3.1) *EDIV*, and *Rand* values shall be set to zero.

If the link is already encrypted then the encryption pause procedure is performed using STK generated in Phase 2 as the Long Term Key provided to the Link Layer (see [Vol 6] Part B, Section 5.1.3.2). *EDIV* and *Rand* values shall be set to zero.

#### *2.4.4.2 Encryption Setup using LTK*

The master device must have the security information (*LTK*, *EDIV*, and *Rand*) distributed by the slave device to setup an encrypted session.

The master initiates the encrypted session using *LTK* distributed by the slave device as the Long Term Key, and the *EDIV* and *Rand* values distributed by the slave device; see [Vol 6] Part B, Section 5.1.3.1.

If the link is already encrypted the encryption pause procedure is performed using LTK distributed by the slave device as the Long Term Key, and the EDIV and Rand values distributed by the slave device; see [Vol 6] Part B, Section 5.1.3.2.

The *EDIV* and *Rand* values are used to establish *LTK* which is used as the Long Term Key on the slave device. If LTK cannot be established from EDIV and Rand values then the slave shall reject the request to encrypt the link and may optionally disconnect the link.

When the security information is stored, subsequent encryption setups may fail if the remote device has deleted the security information. Table 2.5 defines what shall be done depending on the type of the security properties and whether or not bonding was performed when subsequent encryption setup fails.

| Security Properties | Devices Bonded | Action to take when enabling encryption fails |
|---|---|---|
| Unauthenticated, no MITM protection | No | Depends on security policy of the device:<br>• Option 1: Automatically initiate pairing<br>• Option 2: Notify user and ask if pairing is ok.<br>Option 1 is recommended. |
| Unauthenticated, no MITM protection | Yes | Notify user of security failure |
| Authenticated MITM protection | No | Depends on security policy of the device:<br>• Option 1: Automatically initiate pairing<br>• Option 2: Notify user and ask if pairing is ok.<br>Option 2 is recommended. |
| Authenticated MITM protection | Yes | Notify user of security failure |

*Table 2.5: Action after encryption setup failure*

### 2.4.5 Signing Algorithm

An LE device can send signed data without having to establish an encrypted session with a peer device. Data shall be signed using CSRK. A device performing signature verification must have received CSRK from the signing device.

The following are inputs to the signing algorithm:

*m* is variable length

*k* is 128 bits

*SignCounter* is 32 bits

Signing shall be performed using the algorithm defined in the NIST Special Publication 800-38B (http://csrc.nist.gov/publications/PubsSPs.html) using AES-128 as the block cipher. The description of the algorithm can also be found in IETF RFC 4493 (http://www.ietf.org/rfc/rfc4493.txt). NIST SP 800-38B defines the message authentication code (MAC) generation function:

$$MAC = CMAC(K, M, Tlen)$$

The bit length of the MAC (*Tlen*) shall be 64 bits. The key used for signature generation (*k*) shall be set to CSRK.

The message to be signed (*M*) by the CMAC function is the concatenation of the variable length message to be signed (*m*) and 4 octet string representing the 32-bit counter value (*SignCounter*) least significant octet first.

*M = data || SignCounter*

For example, if data to be signed is the 7 octet sequence ‘3456789ABCDEF1’ and *SignCounter* is set to 67653874 (0x040850F2) then *M* is the octet sequence ‘3456789ABCDEF1F2500804’. Examples of CMAC generation using AES-128 as the block cipher are included in NIST Special Publication 800-38B Appendix D and IETF RFC 4493 Section 4.

The *SignCounter* shall be initialized to zero when CSRK is generated and incremented for every message that is signed with a given CSRK.

Note: If a device generates 100,000 signed events a day, a 32-bit counter will wrap after approximately 117 years.

The 64-bit result of the CMAC function is used as the result of the signing algorithm.

To verify a signature a device computes the MAC of a received message and *SignCounter* and compares it with the received MAC. If the MAC does not match then the signature verification has failed. If the MACs match then the signature verification has succeeded.

The device performing verification should store the last verified *SignCounter* in the security database and compare it with a received *SignCounter* to prevent replay attacks. If the received *SignCounter* is greater than the stored value then the message has not been seen by the local device before and the security database can be updated.

### 2.4.6 Slave Security Request

The slave device may request security by transmitting a Security Request command to the master. When a master device receives a Security Request command it may encrypt the link, initiate the pairing procedure, or reject the request.

The slave shall not send the Security Request command if the pairing procedure is in progress, or if the encryption procedure is in progress.

The Security Request command includes the required security properties. A security property of MITM protection required shall only be set if the slave’s IO capabilities would allow the Passkey Entry association model to be used or out of band authentication data is available.

The master shall ignore the slave’s Security Request if the master has sent a Pairing Request without receiving a Pairing Response from the slave or if the master has initiated encryption mode setup.

If pairing or encryption mode is not supported or cannot be initiated at the time when the slave's Security Request Command is received, then the master shall respond with a Pairing Failed Command with the reason set to "Pairing Not Supported."

After receiving a Security Request, the master shall first check whether it has the required security information to enable encryption; see Section 2.4.4.2. If this information is missing or does not meet the security properties requested by the slave, then the master shall initiate the pairing procedure. If the pairing procedure is successful, the master's security database is updated with the keys and security properties are distributed during the pairing procedure.

If the master has the required security information to enable encryption and it meets the security properties request by the slave, it shall perform encryption setup using LTK, see Section 2.4.4.2.

Figure 2.2 shows a summary of the actions and decisions that a master shall take when receiving a Security Request.

The slave shall check that any Pairing Request command received from the master after sending a Security Request command contains Authentication Requirements that meet the requested security properties.

If the slave requests a security property that is not Just Works and receives an encryption procedure request after sending a Security Request command then it shall check that any existing Security Information is of sufficient security properties.

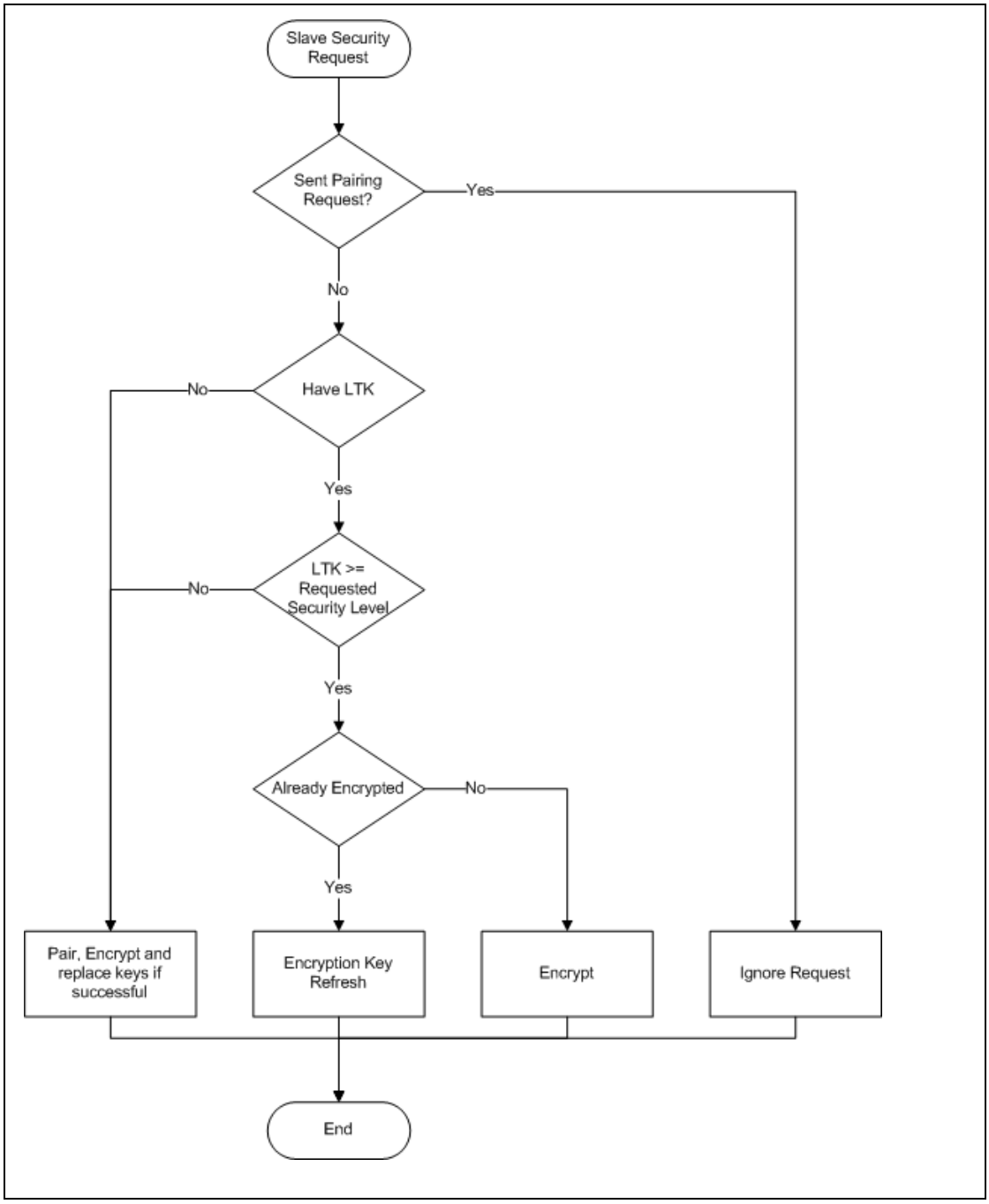



*Figure 2.2: Master actions after receiving Security Request*

# 3 SECURITY MANAGER PROTOCOL

## 3.1 INTRODUCTION

The Security Manager Protocol (SMP) is used for pairing and transport specific key distribution.

## 3.2 SECURITY MANAGER CHANNEL OVER L2CAP

All SMP commands are sent over the Security Manager Channel which is an L2CAP fixed channel (see [Vol 3] Part A, Section 2.1). The configuration parameters for the Security Manager Channel shall be as shown below in Table 3.1.

| Parameter | Value |
|---|---|
| MTU | 23 |
| Flush Timeout | 0xFFFF (Infinite) |
| QoS | Best Effort |
| Mode | Basic Mode |

*Table 3.1: Security Manager Channel Configuration Parameters*

## 3.3 COMMAND FORMAT

The general format for all SMP commands is shown in Figure 3.1.

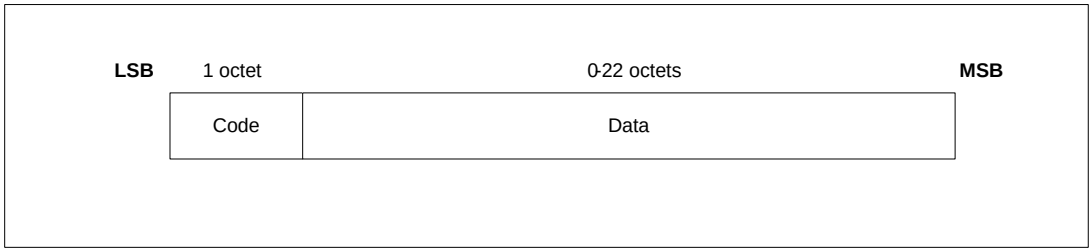



*Figure 3.1: SMP Command Format*

The following are the fields shown:

- *Code (1 octet)*

  The Code field is one octet long and identifies the type of command. Table 3.2 lists the codes defined by this document. If a packet is received with a reserved Code it shall be ignored.

| Code | Description |
|---|---|
| 0x00 | Reserved |
| 0x01 | Pairing Request |
| 0x02 | Pairing Response |
| 0x03 | Pairing Confirm |
| 0x04 | Pairing Random |
| 0x05 | Pairing Failed |
| 0x06 | Encryption Information |
| 0x07 | Master Identification |
| 0x08 | Identity Information |
| 0x09 | Identity Address Information |
| 0x0A | Signing Information |
| 0x0B | Security Request |
| 0x0C – 0xFF | Reserved |

*Table 3.2: SMP Command Codes*

- *Data (0 or more octets)*

  The Data field is variable in length. The Code field determines the format of the Data field.

If a device does not support pairing then it shall respond with a Pairing Failed command with the reason set to “Pairing Not Supported” (see Section 3.5.5) when any command is received. If pairing is supported then all commands shall be supported.

## 3.4 SMP TIMEOUT

To protect the Security Manager protocol from stalling, a Security Manager Timer is used. Upon transmission of the Pairing Request command or reception of the Pairing Request command, the Security Manager Timer shall be reset and started.

The Security Manager Timer shall be reset when an L2CAP SMP command is queued for transmission.

When SMP completes, the Security Manager Timer shall be stopped.

If the Security Manager Timer reaches 30 seconds, the procedure shall be considered to have failed, and the local higher layer shall be notified. No further SMP commands shall be sent over the L2CAP Security Manager Channel. A

new SM procedure shall only be performed when a new physical link has been established.

## 3.5 PAIRING METHODS

The SMP commands defined in this section are used to perform Pairing Feature Exchange and STK Generation (see Section 2.1).

### 3.5.1 Pairing Request

The initiator starts the Pairing Feature Exchange by sending a Pairing Request command to the responding device. The Pairing Request command is defined in Figure 3.2.

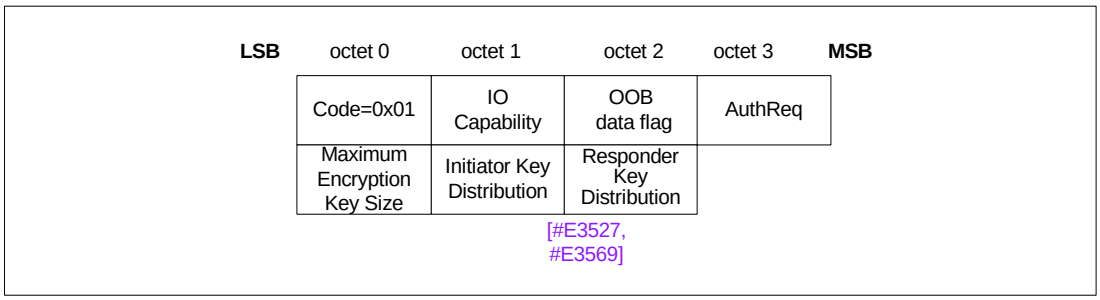



*Figure 3.2: Pairing Request Packet*

The following data fields are used:

- *IO Capability (1 octet)*

  Table 3.3 defines the values which are used when exchanging IO capabilities (see Section 2.3.2).

| Value | Description |
|---|---|
| 0x00 | DisplayOnly |
| 0x01 | DisplayYesNo |
| 0x02 | KeyboardOnly |
| 0x03 | NoInputNoOutput |
| 0x04 | KeyboardDisplay |
| 0x05-0xFF | Reserved |

*Table 3.3: IO Capability Values*

- *OOB data flag (1 octet)*

  Table 3.4 defines the values which are used when indicating whether OOB authentication data is available (see Section 2.3.3).

| Value | Description |
|---|---|
| 0x00 | OOB Authentication data not present |
| 0x01 | OOB Authentication data from remote device present |
| 0x02-0xFF | Reserved |

*Table 3.4: OOB Data Present Values*

- *AuthReq (1 octet)*

  The AuthReq field is a bit field that indicates the requested security properties (see Section 2.3.1) for STK and GAP bonding information (see Part C, Section 9.4).

  Figure 3.3 defines the authentication requirements bit field.

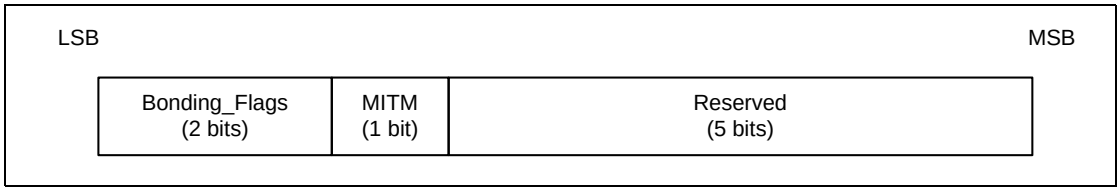



*Figure 3.3: Authentication Requirements Flags*

The Bonding_Flags field is a 2-bit field that indicates the type of bonding being requested by the initiating device as defined in Table 3.5.

| Bonding_Flags $b_1b_0$ | Bonding Type |
|---|---|
| 00 | No Bonding |
| 01 | Bonding |
| 10 | Reserved |
| 11 | Reserved |

*Table 3.5: Bonding Flags*

The MITM field is a 1-bit flag that is set to one if the device is requesting MITM protection, otherwise it shall be set to 0. A device sets the MITM flag to one to request an Authenticated security property for STK.

The reserved 5-bit field shall be set to zero and ignored upon reception.

- *Maximum Encryption Key Size (1 octet)*

  This value defines the maximum encryption key size in octets that the device can support. The maximum key size shall be in the range 7 to 16 octets.

- *Initiator Key Distribution (1 octet)*

  The Initiator Key Distribution field indicates which keys the initiator is requesting to distribute or use during the Transport Specific Key Distribution

phase (see Section 2.4.3). The Initiator Key Distribution field format and usage is defined in Section 3.6.1.

- *Responder Key Distribution (1 octet)*

  The Responder Key Distribution field indicates which keys the initiator is requesting the responder to distribute or use during the Transport Specific Key Distribution phase (see Section 2.4.3). The Responder Key Distribution field format and usage is defined in Section 3.6.1.

### 3.5.2 Pairing Response

This command is used by the responding device to complete the Pairing Feature Exchange after it has received a Pairing Request command from the initiating device, if the responding device allows pairing. The Pairing Response command is defined in Figure 3.4.

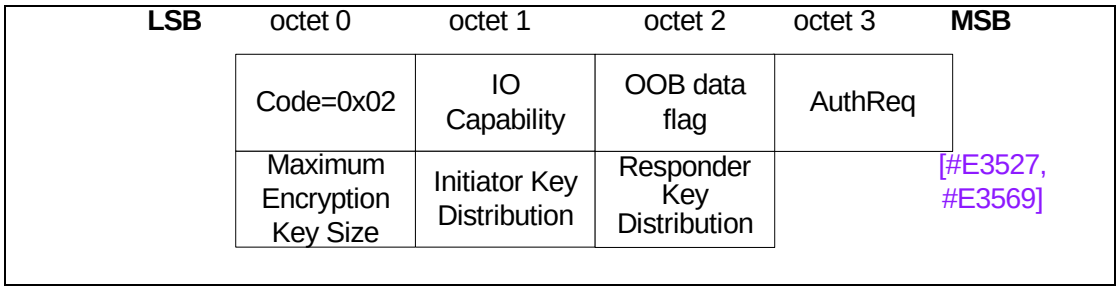



*Figure 3.4: Pairing Response Packet*

The following data fields are used:

- *IO Capability (1 octet)*

  Table 3.3 defines the values which are used when exchanging IO capabilities (see Section 2.3.2).

- *OOB data flag (1 octet)*

  Table 3.4 defines the values which are used when indicating whether OOB authentication data is available (see Section 2.3.3).

- *AuthReq (1 octet)*

  The AuthReq field is a bit field that indicates the requested security properties (see Section 2.3.1) for STK and GAP bonding information (see Part C, Section 9.4).

  Figure 3.3 defines the authentication requirements bit field.

  The Bonding_Flags field is a 2-bit field that indicates the type of bonding being requested by the responding device as defined in Table 3.5.

  The MITM field is a 1-bit flag that is set to one if the device is requesting MITM protection, otherwise it shall be set to 0. A device sets the MITM flag to one to request an Authenticated security property for STK.

  The reserved 5-bit field shall be set to zero and ignored upon reception.

- *Maximum Encryption Key Size (1 octet)*

This value defines the maximum encryption key size in octets that the device can support. The maximum key size shall be in the range 7 to 16 octets.

- *Initiator Key Distribution (1 octet)*

  The Initiator Key Distribution field defines which keys the initiator shall distribute and use during the Transport Specific Key Distribution phase (see Section 2.4.3). The Initiator Key Distribution field format and usage are defined in Section 3.6.1.

- *Responder Key Distribution (1 octet)*

  The Responder Key Distribution field defines which keys the responder shall distribute and use during the Transport Specific Key Distribution phase (see Section 2.4.3). The Responder Key Distribution field format and usage are defined in Section 3.6.1.

### 3.5.3 Pairing Confirm

This is used following a successful Pairing Feature Exchange to start STK Generation. The Pairing Confirm command is defined in Figure 3.5.

This command is used by both devices to send the confirm value to the peer device, see Section 2.3.5.5.

The initiating device starts STK Generation by sending the Pairing Confirm command to the responding device. If the initiating device wants to abort pairing it can transmit a Pairing Failed command instead.

The responding device sends the Pairing Confirm command after it has received a Pairing Confirm command from the initiating device.

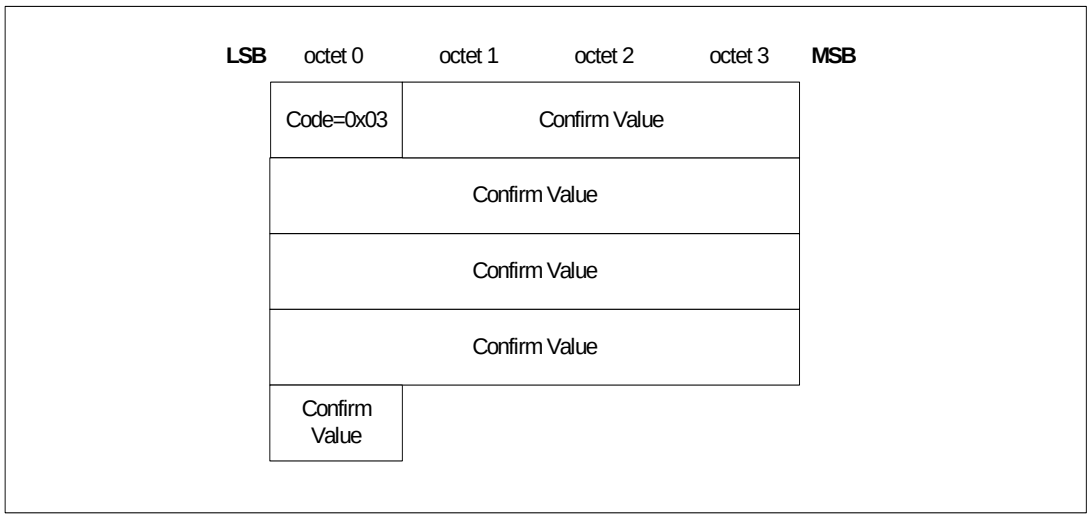



*Figure 3.5: Pairing Confirm Packet*

The following data field is used:

- *Confirm value (16 octets)*

The initiating device sends *Mconfirm* and the responding device sends *Sconfirm* as defined in Section 2.3.5.5.

### 3.5.4 Pairing Random

This command is used by the initiating and responding device to send the random number used to calculate the Confirm value sent in the Pairing Confirm command. The Pairing Random command is defined in Figure 3.6.

The initiating device sends a Pairing Random command after it has received a Pairing Confirm command from the responding device.

The responding device shall send a Pairing Random command after it has received a Pairing Random command from the initiating device if the Confirm value calculated on the responding device matches the Confirm value received from the initiating device. If the calculated Confirm value does not match then the responding device shall respond with the Pairing Failed command.

The initiating device shall encrypt the link using the generated STK if the Confirm value calculated on the initiating device matches the Confirm value received from the responding device. The successful encryption or re-encryption of the link is the signal to the responding device that STK Generation has completed successfully. If the calculated Confirm value does not match then the initiating device shall respond with the Pairing Failed command.

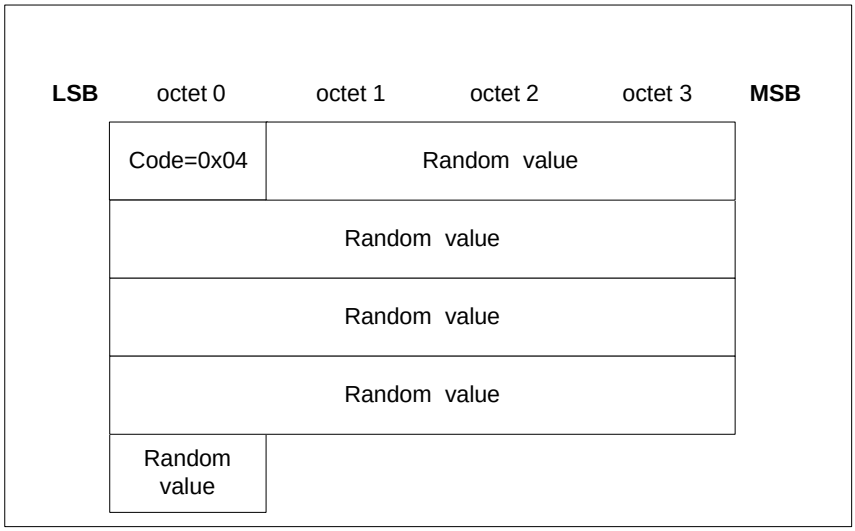



*Figure 3.6: Pairing Random Packet*

The following are the data fields:

- *Random value (16 octets)*

  The initiating device sends *Mrand* and the responding device sends *Srand* as defined in Section 2.3.5.5.

### 3.5.5 Pairing Failed

This is used when there has been a failure during pairing and reports that the pairing procedure has been stopped and no further communication for the current pairing procedure is to occur. The Pairing Failed command is defined in Figure 3.7.

Any subsequent pairing procedure shall restart from the Pairing Feature Exchange phase.

This command may be sent at any time during the pairing process by either device in response to a message from the remote device.

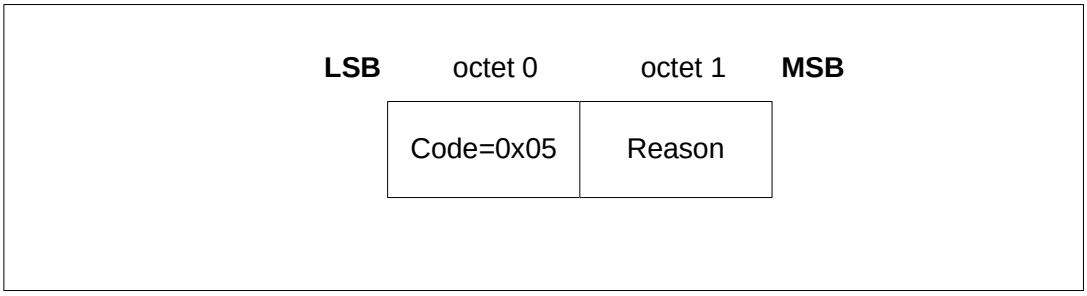



*Figure 3.7: Pairing Failed Packet*

The following data field is used:

- *Reason (1 octets)*

The Reason field indicates why the pairing failed. The reason codes are defined in Table 3.6

| Value | Name | Description |
|---|---|---|
| 0x00 | Reserved | Reserved for future use |
| 0x01 | Passkey Entry Failed | The user input of passkey failed, for example, the user cancelled the operation |
| 0x02 | OOB Not Available | The OOB data is not available |
| 0x03 | Authentication Requirements | The pairing procedure cannot be performed as authentication requirements cannot be met due to IO capabilities of one or both devices |
| 0x04 | Confirm Value Failed | The confirm value does not match the calculated compare value |
| 0x05 | Pairing Not Supported | Pairing is not supported by the device |
| 0x06 | Encryption Key Size | The resultant encryption key size is insufficient for the security requirements of this device |

*Table 3.6: Pairing Failed Reason Codes*

| Value | Name | Description |
|---|---|---|
| 0x07 | Command Not Supported | The SMP command received is not supported on this device |
| 0x08 | Unspecified Reason | Pairing failed due to an unspecified reason |
| 0x09 | Repeated Attempts | Pairing or authentication procedure is disallowed because too little time has elapsed since last pairing request or security request |
| 0x0-0xFF | Invalid Parameters | The Invalid Parameters error code indicates<br>the command length is invalid<br>a parameter is outside of the specified range. |
| 0x0B] | Reserved | Reserved for future use |

*Table 3.6: Pairing Failed Reason Codes*

## 3.6 SECURITY IN BLUETOOTH LOW ENERGY

### 3.6.1 Key Distribution

Bluetooth low energy devices can distribute keys from the slave to the master and from the master to the slave device. The following keys may be distributed from the slave to the master:

- LTK using Encryption Information command
- EDIV and Rand using Master Identification command
- IRK using Identity Information command
- Public device or static random address using Identity Address Information command
- CSRK using Signing Information command

The master may distribute to the slave the following key:

- LTK using Encryption Information command
- EDIV and Rand using Master Identification command
- IRK using Identity Information command
- Public device or static random address using Identity Address Information command
- CSRK using Signing Information command

The keys which are to be distributed in the Transport Specific Key Distribution phase are indicated in the Key Distribution field of the Pairing Request and Pairing Response commands see Section 3.5.1 and Section 3.5.2.

The format of the Initiator Key Distribution field and Responder Key Distribution field in the Pairing Request and Pairing Response commands for LE is defined in Figure 3.8.

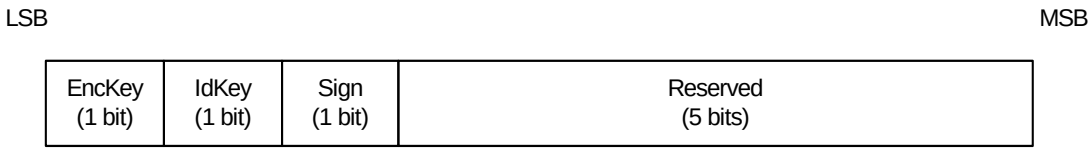



*Figure 3.8: LE Key Distribution Format*

The Key Distribution field has the following flags:

- EncKey is a 1-bit field that is set to one to indicate that the device shall distribute LTK using the Encryption Information command followed by EDIV and Rand using the Master Identification command.
- IdKey is a 1-bit field that is set to one to indicate that the device shall distribute IRK using the Identity Information command followed by its public device or static random address using Identity Address Information.
- Sign is a 1-bit field that is set to one to indicate that the device shall distribute CSRK using the Signing Information command.
- Reserved is a 5-bit field that shall be set to zero and ignored on reception.

The Initiator Key Distribution field in the Pairing Request command is used by the master to request which keys are distributed by the initiator to the responder. The Responder Key Distribution field in the Pairing Request command is used by the master to request which keys are distributed by the responder to the initiator. The Initiator Key Distribution field in the Pairing Response command from the slave defines the keys that shall be distributed by the initiator to the responder. The Responder Key Distribution field in the Pairing Response command from the slave defines the keys that shall be distributed by the responder to the initiator. The slave shall not set to one any flag in the Initiator Key Distribution or Responder Key Distribution field of the Pairing Response command that the master has set to zero in the Initiator Key Distribution and Responder Key Distribution fields of the Pairing Request command.

The keys shall be distributed in the following order:

1. LTK by the slave
2. EDIV and Rand by the slave
3. IRK by the slave
4. BD ADDR by the slave
5. CSRK by the slave
6. LTK by the master
7. EDIV and Rand by the master

8. IRK by the master
9. BD_ADDR by the master
10. CSRK by the master

If a key is not being distributed then the command to distribute that key shall not be sent.

Note: If a key is not distributed, then the capabilities that use this key will not be available. For example, if a LTK is not distributed from the slave to the master, then the master cannot encrypt a future link with that slave, therefore pairing would have to be performed again.

Note: The initiator should determine the keys needed based on the capabilities that are required by higher layer specifications. For example, if the initiator determines that encryption is required in a future link with that slave, then the initiator must request that slave's LTK is distributed by setting the EncKey bit to one in the Responder Key Distribution field of the Pairing Request command.

If EncKey, IdKey, and Sign are set to zero in the Initiator Key Distribution and Responder Key Distribution fields, then no keys shall be distributed and the link will be encrypted using the generated STK.

Key distribution is complete in the device sending the final key when it receives the baseband acknowledgement for that key and is complete in the receiving device when it receives the final key being distributed.

### 3.6.2 Encryption Information

[Encryption Information is used in the Transport Specific Key Distribution to distribute LTK that is used when encrypting future connections. The Encryption Information command is defined in Figure 3.9.

The Encryption Information command shall only be sent when the link has been encrypted or re-encrypted using the generated STK.

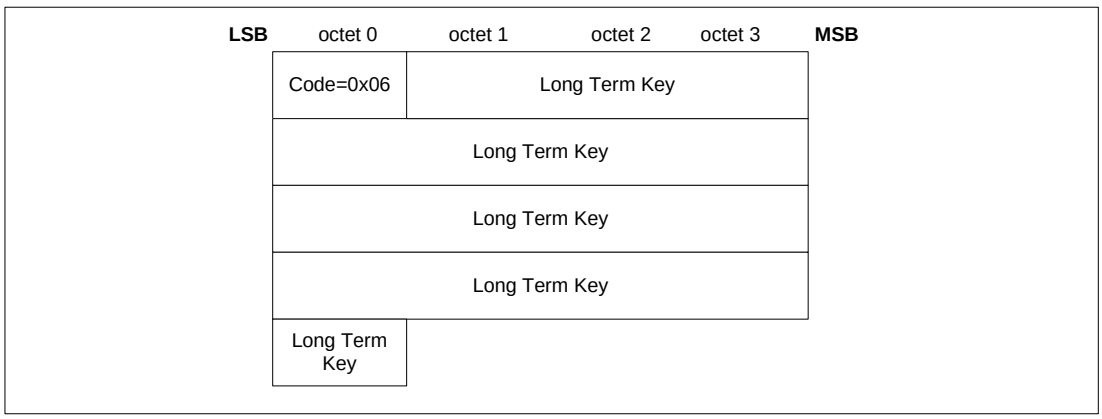



*Figure 3.9: Encryption Information Packet*

The following is the data field:

- *Long Term Key (16 octets)*

  The generated LTK value being distributed, see Section 2.4.2.3.

### 3.6.3 Master Identification

Master Identification is used in the Transport Specific Key Distribution phase to distribute EDIV and Rand which are used when encrypting future connections. The Master Identification command is defined in Figure 3.10.

The Master Identification command shall only be sent when the link has been encrypted or re-encrypted using the generated STK.

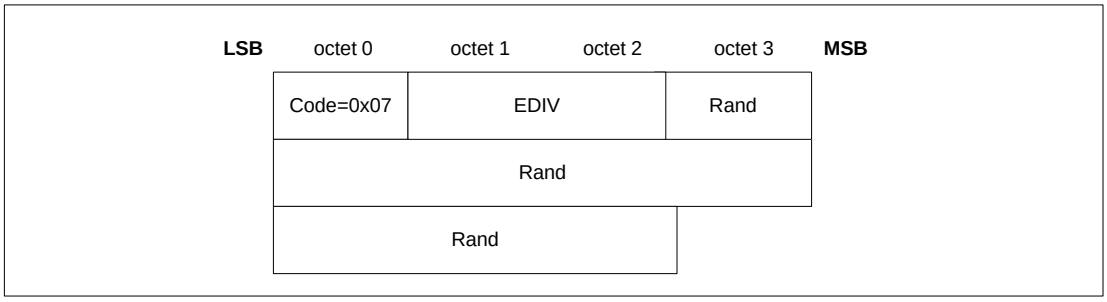



*Figure 3.10: Master Identification Packet*

The following data fields are used:

- *EDIV (2 octets)*

  The EDIV value being distributed (see Section 2.4.2.3).

- *Rand (8 octets)*

  64-bit Rand value being distributed (see Section 2.4.2.3).

### 3.6.4 Identity Information

Identity Information is used in the Transport Specific Key Distribution phase to distribute the IRK. The Identity Information command is defined in Figure 3.11.

The Identity Information command shall only be sent when the link has been encrypted or re-encrypted using the generated STK.

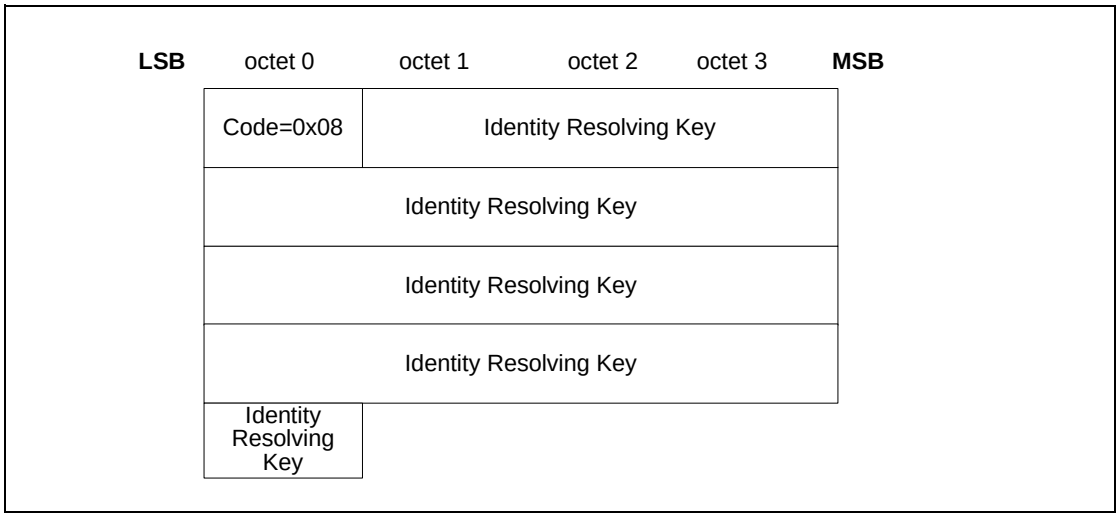



*Figure 3.11: Identity Information Packet*

The following are the data fields:

- *Identity Resolving Key (16 octets)*

  128-bit IRK value being distributed (see Section 2.4.2.1).

### 3.6.5 Identity Address Information

Identity Address Information is used in the Transport Specific Key Distribution phase to distribute its public device address or static random address. The Identity Address Information command is defined in Figure 3.12.

The Identity Address Information command shall only be sent when the link has been encrypted or re-encrypted using the generated STK.

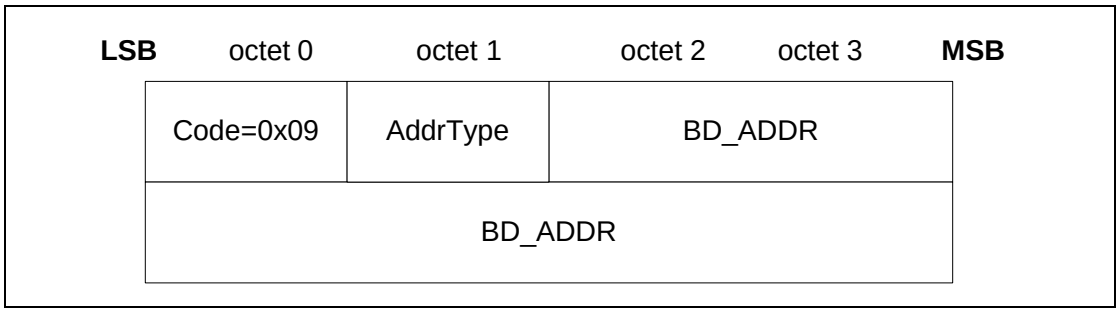



*Figure 3.12: Identity Address Information Packet*

The data fields are:

- *AddrType (1 octet)*

  If BD_ADDR is a public device address or set to all zeros, then AddrType shall be set to 0x00. If BD_ADDR is a static random device address then AddrType shall be set to 0x01.

- *BD_ADDR (6 octets)*

If the distributing device has a public device address or a static random address it shall set this field to its public device address or static random address otherwise it shall set this field to all zeros.

### 3.6.6 Signing Information

Signing Information is used in the Transport Specific Key Distribution to distribute the CSRK which a device uses to sign data. The Signing Information command is defined in Figure 3.13.

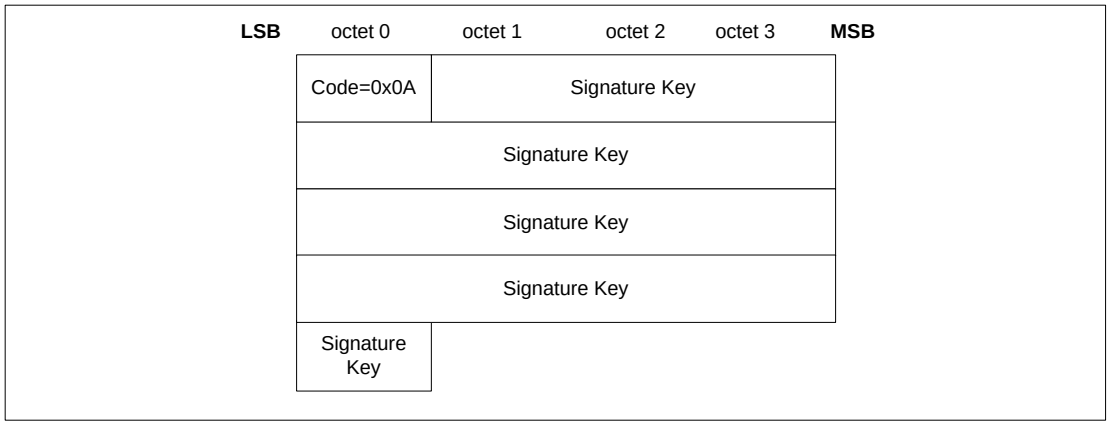



*Figure 3.13: Signing Information Packet*

The following data field is used:

- *Signature Key (16 octets)*

  128-bit CSRK that is being distributed; see Section 2.4.2.2.

### 3.6.7 Security Request

The Security Request command is used by the slave to request that the master initiates security with the requested security properties, see Section 2.4.6. The Security Request command is defined in Figure 3.14.

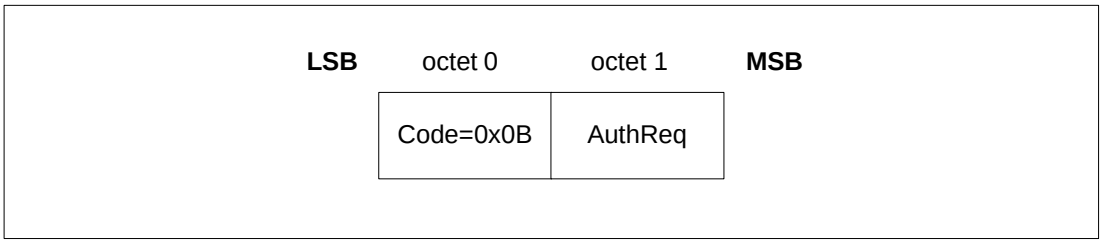



*Figure 3.14: Security Request Packet*

The following data field is used:

- *AuthReq (1 octet)*

The AuthReq field is a bit field that indicates the requested security properties (see Section 2.3.1) for STK and GAP bonding information (see Part C, Section 9.4).

Figure 3.3 defines the authentication requirements bit field.

The Bonding_Flags field is a 2-bit field that indicates the type of bonding being requested by the responding device as defined in Table 3.5.

The MITM field is a 1-bit flag that is set to one if the device is requesting MITM protection, otherwise it shall be set to 0.

The reserved 5-bit field shall be set to zero and ignored upon reception.

# 4 REFERENCES

[1] NIST Publication FIPS-197 (http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf)

# FIGURES

Figure 1.1: Relationship of the Security Manager to the rest of the LE Bluetooth architecture ..... 598
Figure 2.1: LE Pairing Phases ..... 599
Figure 2.2: Master actions after receiving Security Request ..... 619
Figure 3.1: SMP Command Format ..... 620
Figure 3.2: Pairing Request Packet ..... 622
Figure 3.3: Authentication Requirements Flags ..... 623
Figure 3.4: Pairing Response Packet ..... 624
Figure 3.5: Pairing Confirm Packet ..... 625
Figure 3.6: Pairing Random Packet ..... 626
Figure 3.7: Pairing Failed Packet ..... 627
Figure 3.8: LE Key Distribution Format ..... 629
Figure 3.9: Encryption Information Packet ..... 630
Figure 3.10: Master Identification Packet ..... 631
Figure 3.11: Identity Information Packet ..... 632
Figure 3.12: Identity Address Information Packet ..... 632
Figure 3.13: Signing Information Packet ..... 633
Figure 3.14: Security Request Packet ..... 633
Figure 5.1: Example Key Hierarchy ..... 640
Figure 5.2: Pairing Process Overview ..... 643
Figure 5.3: Pairing initiated by master ..... 644
Figure 5.4: Slave security request, master initiated pairing ..... 644
Figure 5.5: Just Works Pairing Method ..... 645
Figure 5.6: Passkey Entry Pairing Method ..... 646
Figure 5.7: OOB Pairing Method ..... 647
Figure 5.8: Transport Specific Key Distribution ..... 648
Figure 5.9: Slave security request, master initiates Link Layer encryption ..... 649
Figure 5.10: Slave rejects pairing attempt ..... 649
Figure 5.11: Master rejects pairing because of key size ..... 650
Figure 5.12: Slave rejects pairing because of key size ..... 650
Figure 5.13: Passkey Entry failure on master ..... 651
Figure 5.14: Passkey Entry failure on slave ..... 651
Figure 5.15: Different Passkeys entered ..... 652
Figure 5.16: Master rejects Sconfirm value from slave ..... 653

# TABLES

Table 2.1: User Input Capabilities ..... 605
Table 2.2: User Output Capabilities ..... 605
Table 2.3: I/O Capabilities Mapping ..... 606
Table 2.4: Mapping of IO Capabilities to STK Generation Method ..... 607
Table 2.5: Action after encryption setup failure ..... 616
Table 3.1: Security Manager Channel Configuration Parameters ..... 620
Table 3.2: SMP Command Codes ..... 621
Table 3.3: IO Capability Values ..... 622
Table 3.4: OOB Data Present Values ..... 623
Table 3.5: Bonding Flags ..... 623
Table 3.6: Pairing Failed Reason Codes ..... 627

# 5 APPENDICES

## 5.1 APPENDIX A – EDIV AND RAND GENERATION

EDIV and Rand are used by the responding device to identify an initiator and recover LTK. This section provides an example of how the distributed EDIV value is a masked version of the real value (DIV) which is used to recover LTK. Other methods can be used that provide equal or higher levels of confidentially for DIV.

### 5.1.1 EDIV Masking

The masking process uses a Diversifier Hiding Key (DHK) which is a 128-bit key that is never distributed.

DHK can be assigned, randomly generated by the device during manufacturing, part of a key hierarchy (see Section 5.2.2.3) or some other method could be used, that results in DHK having 128 bits of entropy. If DHK is randomly generated then the requirements for random generation defined in [Vol 2] Part H, Section 2 shall be used.

If DHK is changed then DIV values cannot be recovered from previously distributed EDIV values.

Section 5.1.1.1 defines a cryptographic function that is used by the responding device when generating EDIV and recovering DIV.

Section 5.1.1.2 describes how a responding device generates an EDIV value to be distributed to an initiating device and Section 5.1.1.3 describes how the responding device recovers DIV from a distributed EDIV value.

#### *5.1.1.1 DIV Mask generation function* dm

DIV is masked before distribution and unmasked during the encryption session setup using the output of the DIV mask generation function *dm*.

The following are inputs to the DIV mask generation function *dm*:

k is 128 bits

r is 64 bits

padding is 64 bits

*r* is concatenated with padding to generate *r*' which is used as the 128-bit input parameter *plaintextData* to security function *e*:

r' = padding || r

The least significant octet of *r* becomes the least significant octet of *r'* and the most significant octet of *padding* becomes the most significant octet of *r'*.

For example, if the 64-bit value *r* is 0x123456789ABCDEF0 then *r'* is 0x0000000000000000123456789ABCDEF0.

The output of the DIV mask generation function *dm* is

$$dm(k, r) = e(k, r') \bmod 2^{16}$$

The output of the security function e is then truncated to 16 bits by taking the least significant 16 bits of the output of *e* as the result of *dm*.

#### *5.1.1.2 EDIV Generation*

The responding device generates a 64-bit random value, *Rand.* The *Rand* value is used to generate 16-bit Y using the DIV mask generation function *dm* with the input parameter *k* set to DHK and the input parameter *r* set to *Rand*.

$$Y = dm(DHK, Rand)$$

The responding device then masks the DIV value to be distributed by bitwise XORing it with Y to generate EDIV.

$$EDIV = Y \text{ xor } DIV$$

EDIV and Rand are distributed to an initiating device during the transport specific key distribution phase using the Master Identification command.

#### *5.1.1.3 DIV Recovery*

When the responding device receives a request to encrypt a session it calculates Y using the DIV mask generation function *dm* with the input parameter *k* set to DHK and the input parameter *r* set to *Rand.* The Y value is bitwise XORed with *EDIV* from the initiator to recover DIV.

$$DIV = Y \text{ xor } EDIV$$

The recovered DIV value can then used to recover LTK which is used to enable encryption on the link.

## 5.2 APPENDIX B – KEY MANAGEMENT

The security provided by different methods can vary and care should be taken to ensure that a chosen method is suitable for a device's requirements.

Section 5.2.1 uses a database for managing the keys. Section 5.2.2 uses a key hierarchy to manage the keys.

### 5.2.1 Database Lookup

The LTK which is distributed is a 128-bit random number which is stored in a database, using EDIV as an index. There is no direct relationship between LTK and EDIV.

The requirements for random generation defined in [Vol 2] Part H, Section 2 shall be used when generating LTK. This method provides an LTK with 128 bits of entropy.

CSRK, IRK, and other keys shall also be stored in the database. There is no relationship between the keys stored in the database or distributed LTKs, EDIVs, or Rands.

If the example EDIV and Rand generation method described in Section 5.1.1 is used then the database shall be used to store DHK. DIV should be used as the index to recover LTK.

### 5.2.2 Key Hierarchy

A key hierarchy can be used to generate the keys.

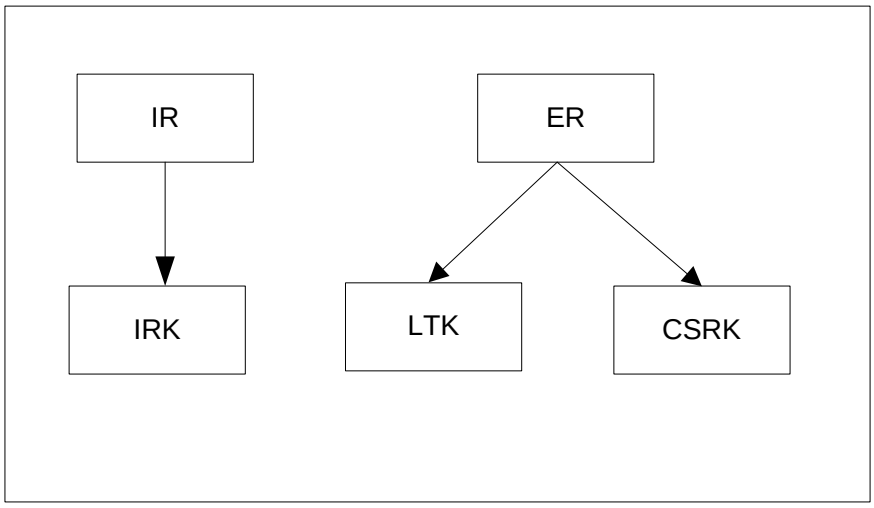



*Figure 5.1: Example Key Hierarchy*

Figure 5.1 is an example key hierarchy where LTK and CSRK are generated from a common ER key, and IRK is generated from a common IR key.

1. ER is a 128-bit key generated for each LE device that supports encrypted connections. It is used to generate LTK using EDIV; see Section 5.2.2.2.
2. IR is a 128-bit key generated for each LE device that supports encrypted connections, uses random addresses or signing data. IR is used to generate IRK and CSRK, see Section 5.2.2.3. It can also be used to generate DHK; see Section 5.1.

New LTK, EDIV, Rand, and CSRK values shall be generated each time they are distributed. If ER is changed then any previously distributed LTK or CSRK keys will no longer be valid.

The distributed IRK shall be the same for all devices it is distributed to. If IR is changed then any previously distributed IRK keys will no longer be valid.

The distributing device only needs to store IR and ER. LTK, IRK, and CSRK can be regenerated when they are required. This reduces the storage requirements on the distributing device.

Section 5.2.2.1 defines an example of the key diversifying function which can be used to generate LTK, IRK, CSRK, and other keys. Other implementations of this function can be used depending upon the exact security requirements of the device.

The NIST Special Publication 800-108 (http://csrc.nist.gov/publications/PubsSPs.html) defines key derivation functions which could be used instead of the example diversifying function *d1*.

### 5.2.2.1 *Diversifying function* d1

Diversified keys are generated with function *d1*. The diversifying function *d1* makes use of the security function *e*.

The following are inputs to diversifying function *d1*:

k is 128 bits

d is 16 bits

r is 16 bits

padding is 96 bits

*d* is concatenated with *r* and *padding* to generate *d',* which is used as the 128-bit input parameter *plaintextData* to security function *e*:

d' = padding || r || d

The least significant octet of *d* becomes the least significant octet of *d'* and the most significant octet of *padding* becomes the most significant octet of *d'*.

For example, if the 16-bit value *d* is 0x1234 and the 16-bit value *r* is 0xabcd, then *d'* is 0x000000000000000000000000abcd1234.

The output diversifying function *d1* is:

*d1*(k, d, r) = e(k, d')

The 128-bit output of the security function *e* is used as the result of diversifying function *d1*.

### 5.2.2.2 Generating Keys from ER

ER is used to generate LTK and CSRK. ER can be assigned, randomly generated by the device during manufacturing or some other method could be used, that results in ER having 128 bits of entropy. If ER is randomly generated then the requirements for random generation defined in [Vol 2] Part H, Section 2 shall be used.

The EDIV and Rand generation method described in Section 5.1 shall be used. LTK is the result of the diversifying function *d1* with the ER as the input parameter k, the DIV as the input parameter d, and the value '0' as the input parameter r; see Section 5.2.2.1.

LTK = *d1*(ER, DIV, 0)

LTK can be recovered from ER and DIV by repeating the calculation when LTK is required.

CSRK is the result of the diversifying function *d1* with the ER as the input parameter k, the DIV as the input parameter d, and the value '1' as the input parameter r; see Section 5.2.2.1.

CSRK = *d1*(ER, DIV, 1)

CSRK can be recovered from ER and DIV by repeating the calculation when CSRK is required.

This method provides an LTK and CSRK with limited amount of entropy because LTK and CSRK are directly related to EDIV and may not be as secure as other generation methods.

To reduce the probability of the same LTK or CSRK value being generated, the DIV values must be unique for each CSRK, LTK, EDIV, and Rand set that is distributed.

A method for preventing a malicious device from repeatedly pairing and collecting CSRK, LTK and DIV information, which could be used in a known plain text attack in ER, should be implemented.

### 5.2.2.3 Generating Keys from IR

IR can be used to generate IRK and other required keys. IR can be assigned, randomly generated by the device during manufacturing or some other method could be used, that results in IR having 128 bits of entropy. If IR is randomly generated then the requirements for random generation defined in [Vol 2] Part H, Section 2 shall be used.

IRK is the result of the diversifying function *d1* with IR as the input parameter k and the value ‘1’ as the input parameter *d* and the value ‘0’ as the input parameter r; see Section 5.2.2.1.

IRK = *d1*(IR, 1, 0)

If the example EDIV and Rand generation method described in Section 5.1.1 is used then DHK can be the result of diversifying function *d1* with IR as the input parameter k and the value ‘3’ as the input parameter *d* and the value ‘0’ as the input parameter r.

DHK = *d1*(IR, 3, 0)

Other keys can be generated by using different values for *k* as the input to the diversifying function *d1*. If the value of *k* is reused for a given IR then the resulting key will be the same.

## 5.3 MESSAGE SEQUENCE CHARTS

A flow diagram of pairing is shown in Figure 5.2. The process has 4 steps. Step 2 has a number of different options.

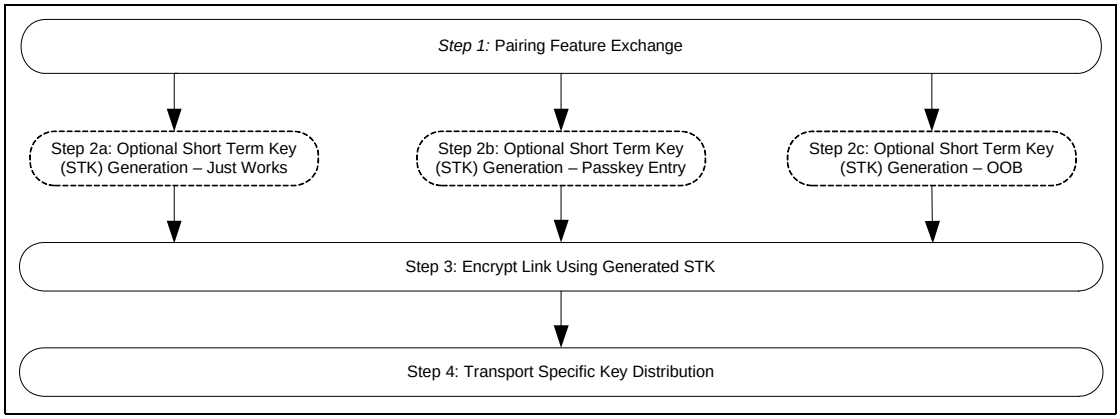



*Figure 5.2: Pairing Process Overview*

### 5.3.1 Phase 1: Pairing Feature Exchange

The master initiates the pairing procedure using Pairing Request command as shown in Figure 5.3.

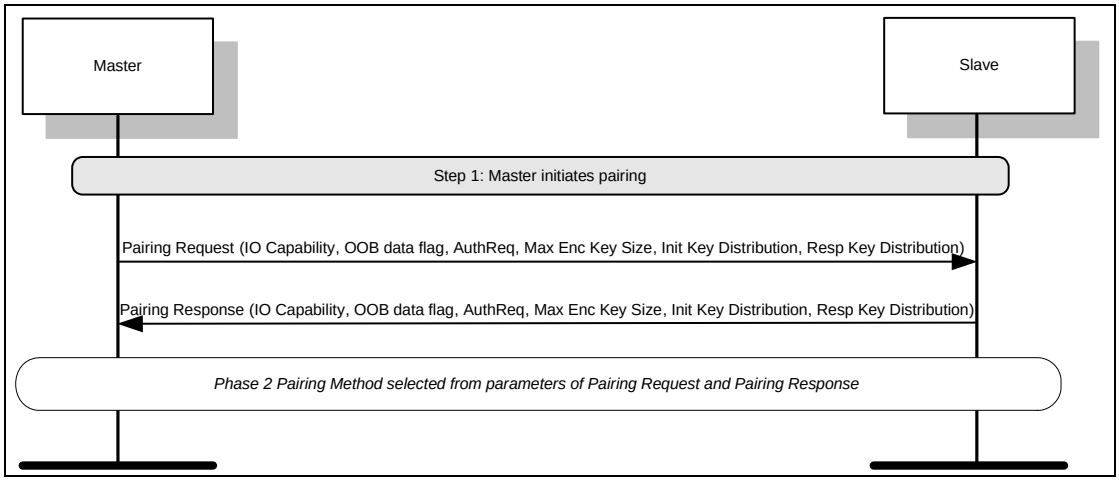



*Figure 5.3: Pairing initiated by master*

#### *5.3.1.1 Slave Security Request – Master Requests Pairing*

The slave may request the master initiates security procedures. Figure 5.4 shows an example where the slave requests security and the master initiates pairing in response.

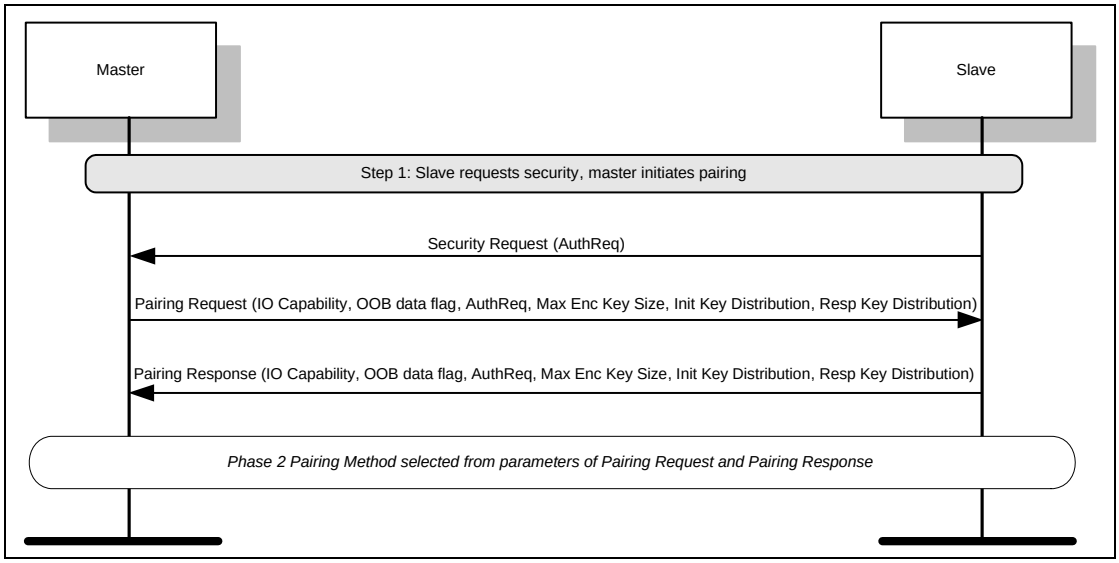



*Figure 5.4: Slave security request, master initiated pairing*

### 5.3.2 Phase 2: Authenticating and Encrypting

After Pairing Feature Exchange has completed a pairing method is selected one of the possible short term key generation sequences are used. This can be Just Works, Passkey Entry or Out of Band pairing method.

### 5.3.2.1 Phase 2: Short Term Key Generation – Just Works

After Pairing Feature Exchange has completed a pairing method is selected. Figure 5.5 shows the Just Works pairing method.

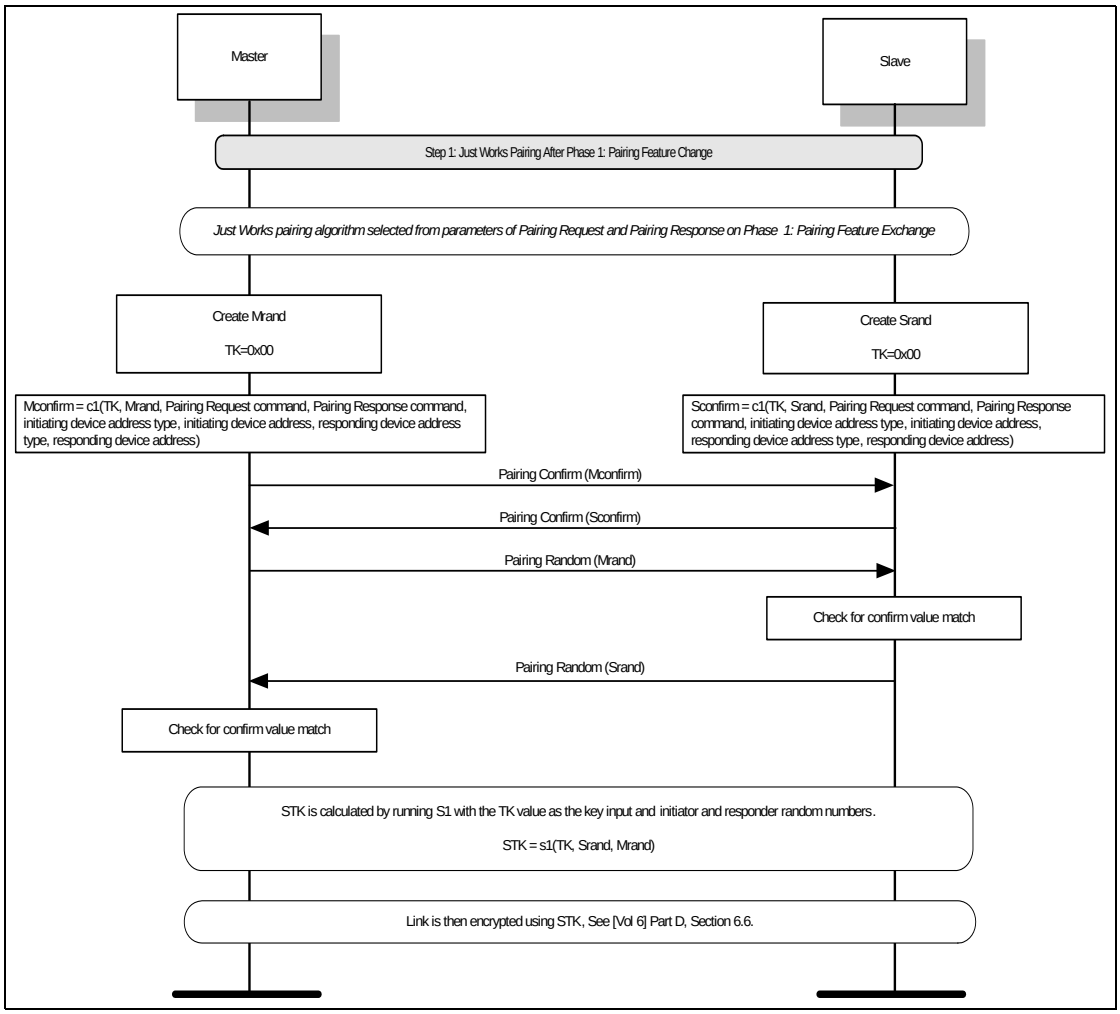



*Figure 5.5: Just Works Pairing Method*

### 5.3.2.2 Phase 2: Short Term Key Generation – Passkey Entry

After Pairing Feature Exchange has completed, a pairing method is selected. Figure 5.6 shows the Passkey Entry pairing method.

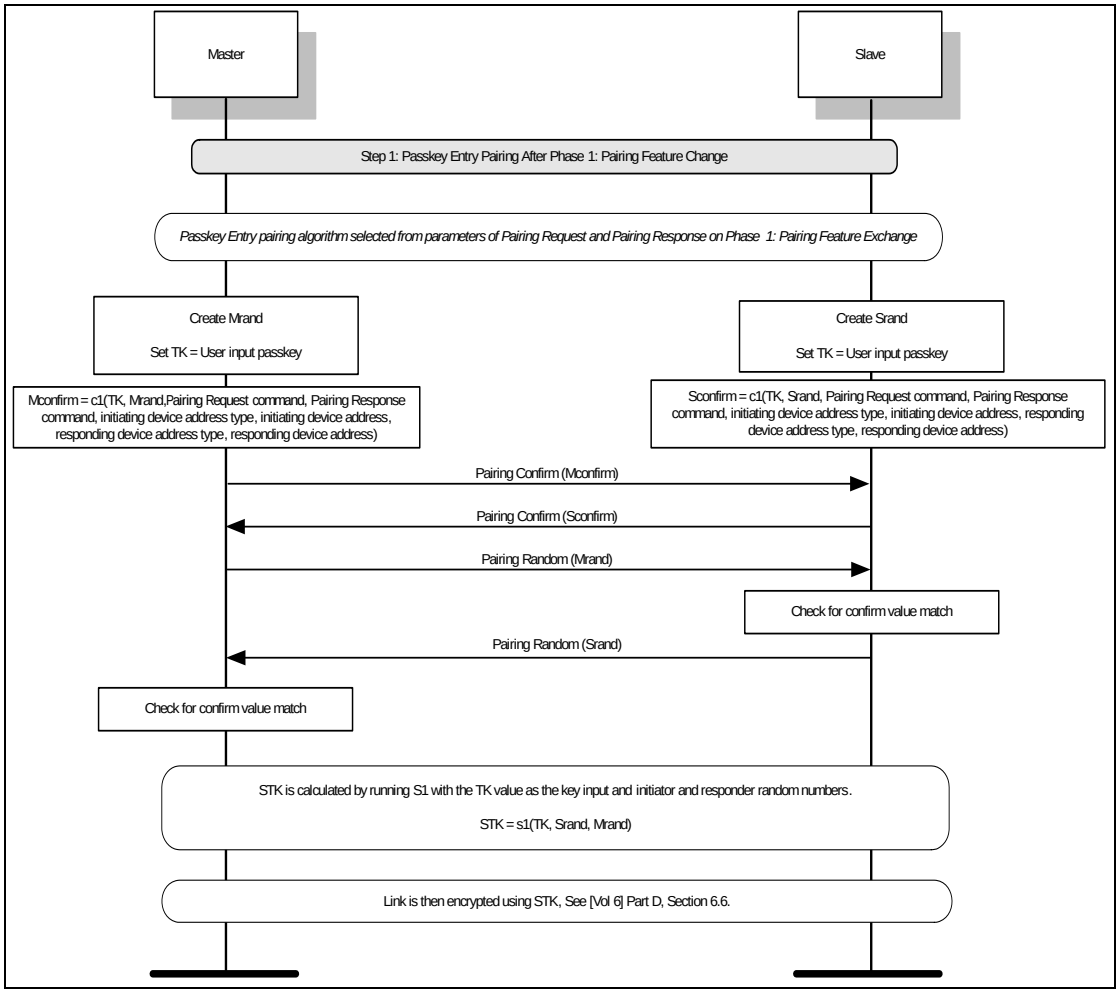



*Figure 5.6: Passkey Entry Pairing Method*

### *5.3.2.3 Phase 2: Short Term Key Generation – Out of Band*

After Pairing Feature Exchange has completed, a pairing method is selected. Figure 5.7 shows the Out of Band pairing method.

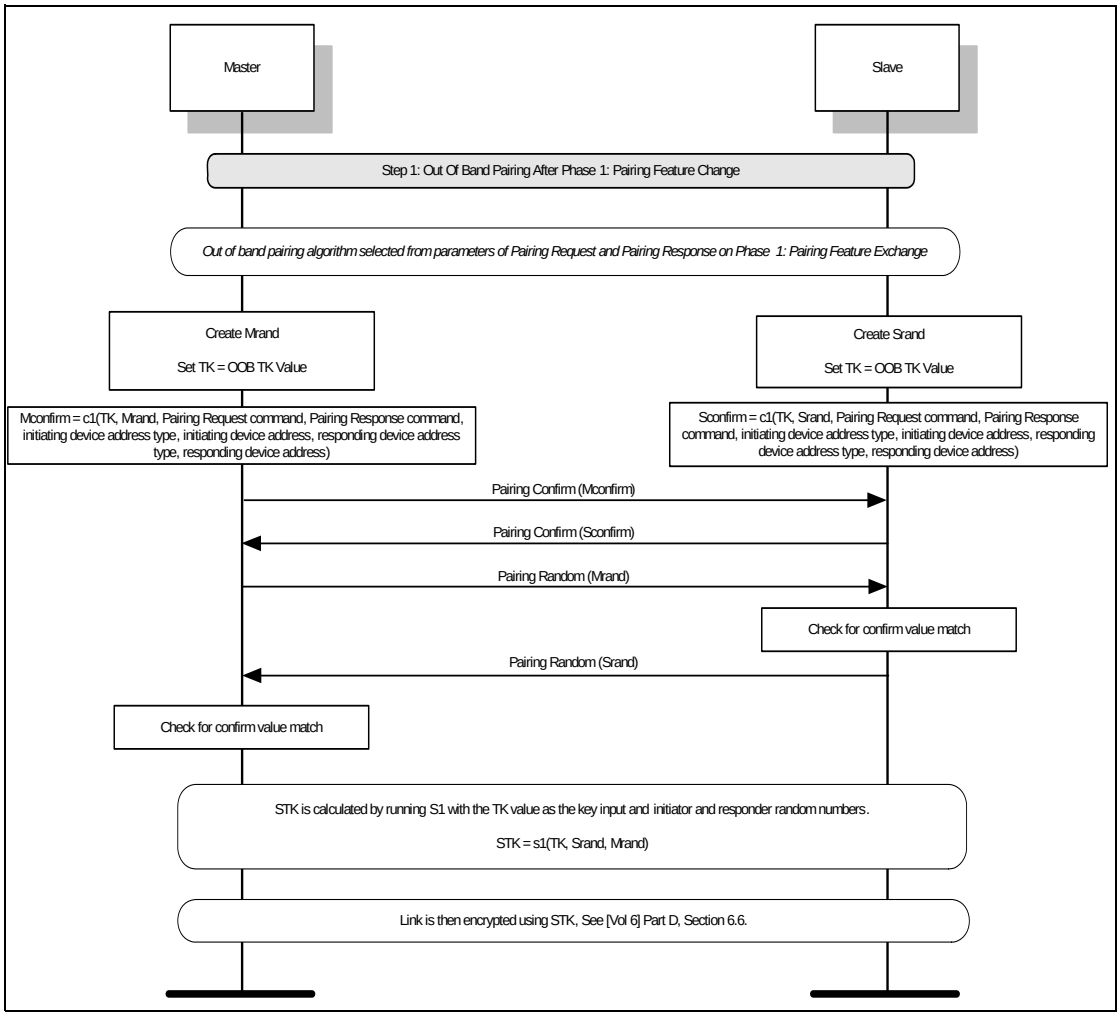



*Figure 5.7: OOB Pairing Method*

### 5.3.3 Phase 3: Transport Specific Key Distribution

After short term key generation and the link has been encrypted, transport specific keys are distributed. Figure 5.8 shows an example of all keys and values being distributed by Master and Slave.

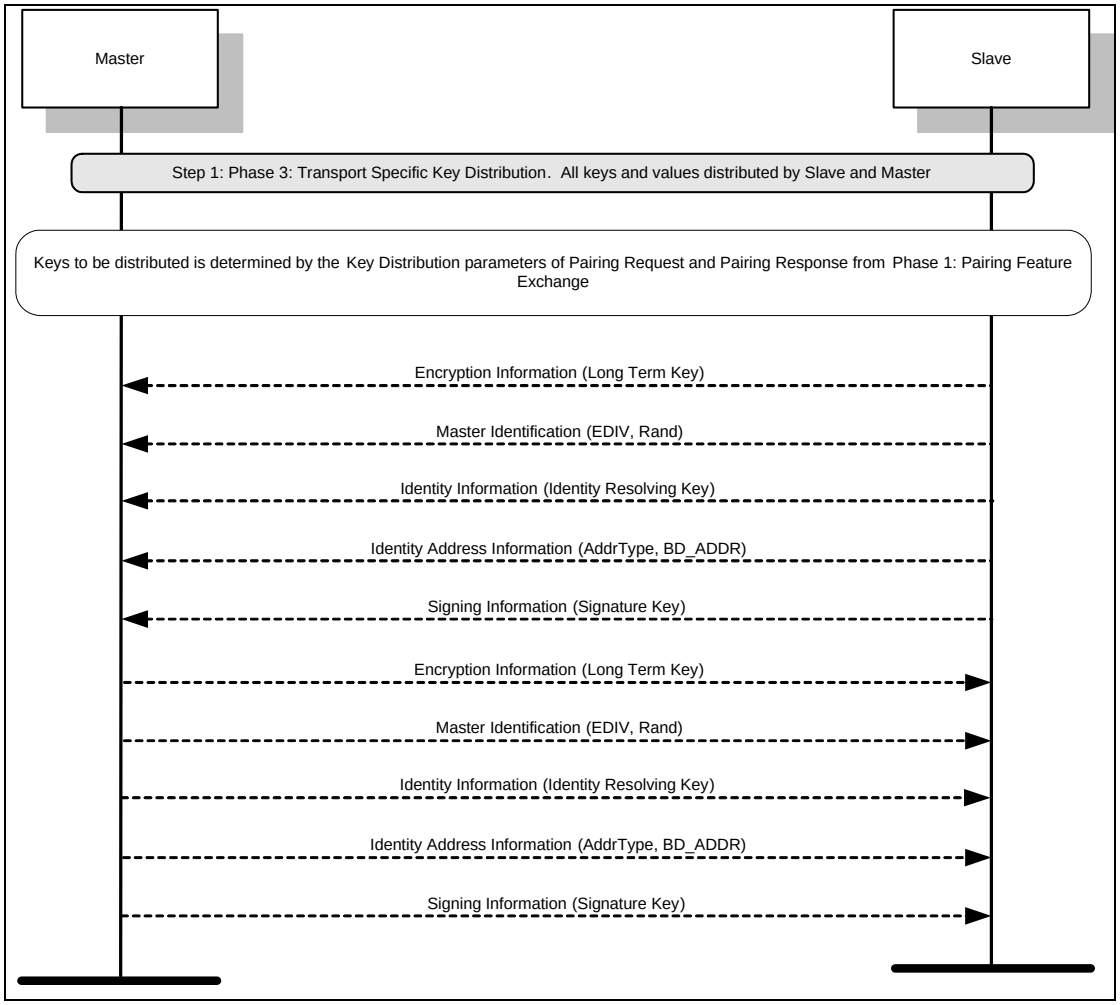



*Figure 5.8: Transport Specific Key Distribution*

### 5.3.4 Security Re-established using Previously Distributed LTK

Devices may re-establish security using a previously distributed LTK. The master device always initiates the encryption procedures, and therefore there are two possible sequences: master initiated and slave requested.

#### *5.3.4.1 Master Initiated Security - Master Initiated Link Layer Encryption*

The master initiates encryption procedures. There is no SM signaling to enable this; the master initiates Link Layer encryption only. See [Vol 6] Part D, Section 6.6.

#### *5.3.4.2 Slave Security Request- Master Initiated Link Layer Encryption*

The slave may request the master initiates security procedures. Figure 5.9 shows an example where the slave requests security and the master initiates Link Layer encryption.

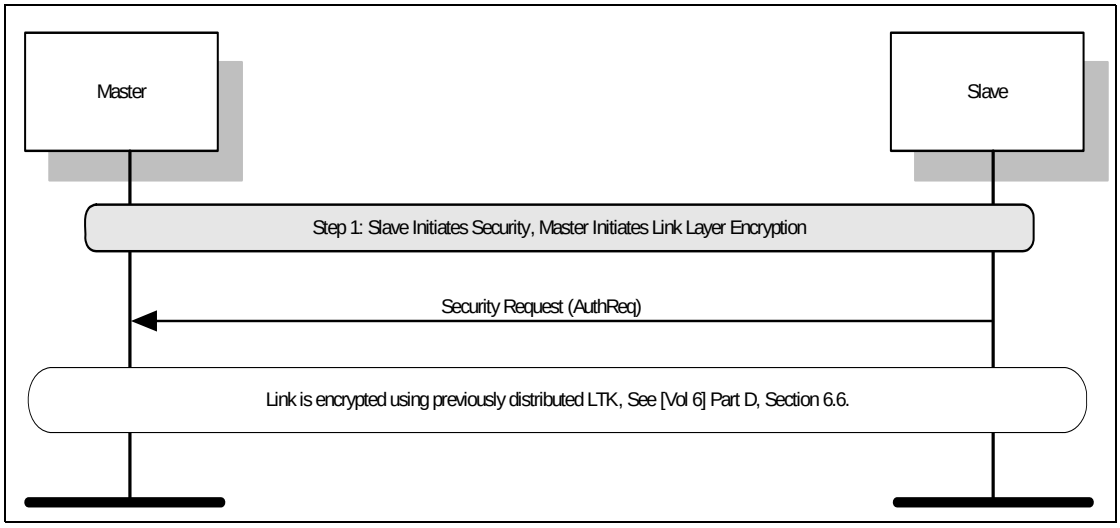



*Figure 5.9: Slave security request, master initiates Link Layer encryption*

### 5.3.5 Failure Conditions

The following sequences show possible failure conditions and their associated signaling.

#### *5.3.5.1 Pairing Not Supported by Slave*

If the slave device does not support pairing or pairing cannot be performed the slave can reject the request from the master. Figure 5.10 shows the slave rejecting a Pairing Request command from the master.

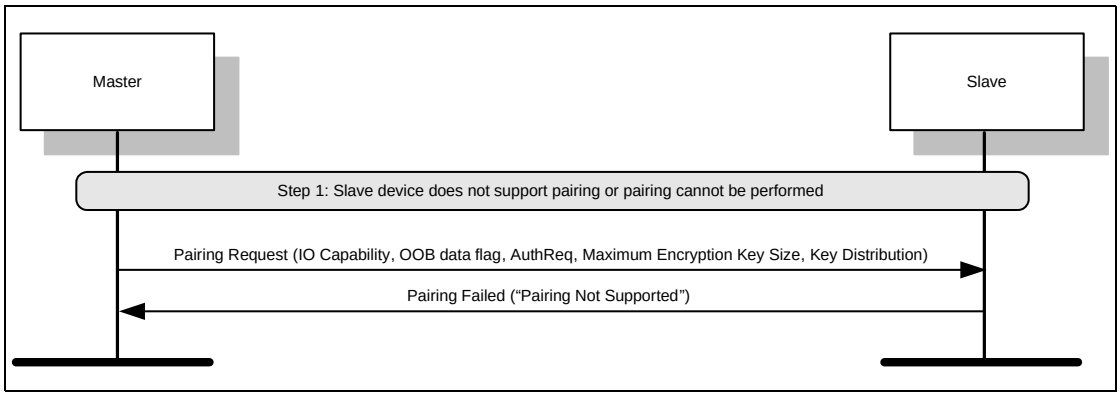



*Figure 5.10: Slave rejects pairing attempt*

#### *5.3.5.2 Master Rejects Pairing Because of Key Size*

During Pairing Feature Exchange the size of the Encryption Key is negotiated. Figure 5.11 shows an example where the master terminates the pairing procedure because the resulting key size is not acceptable.

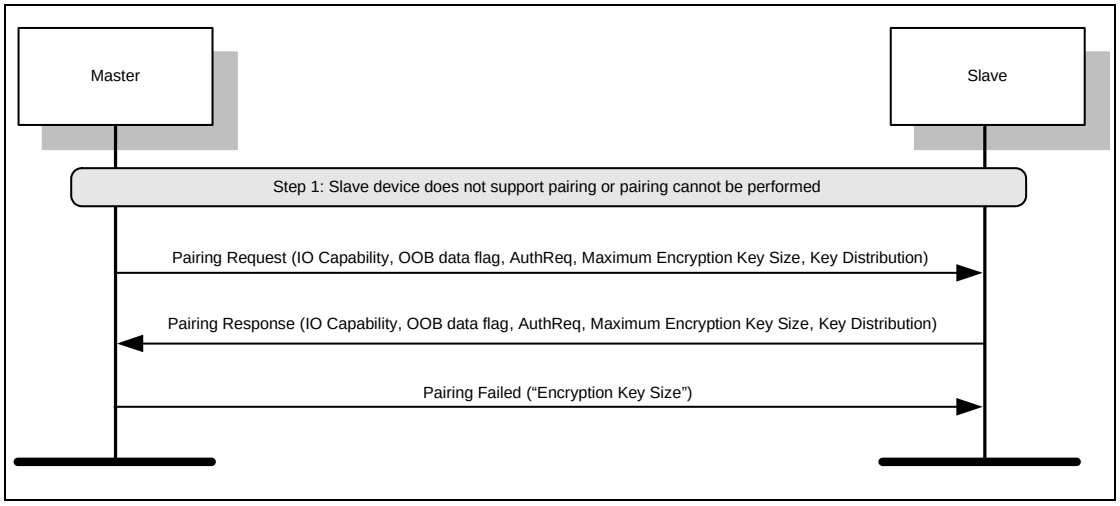



*Figure 5.11: Master rejects pairing because of key size*

### 5.3.5.3 Slave Rejects Pairing Because of Key Size

During Pairing Feature Exchange the size of the Encryption Key is negotiated. Figure 5.12 shows an example where the slave terminates the pairing procedure because the resulting key size is not acceptable.

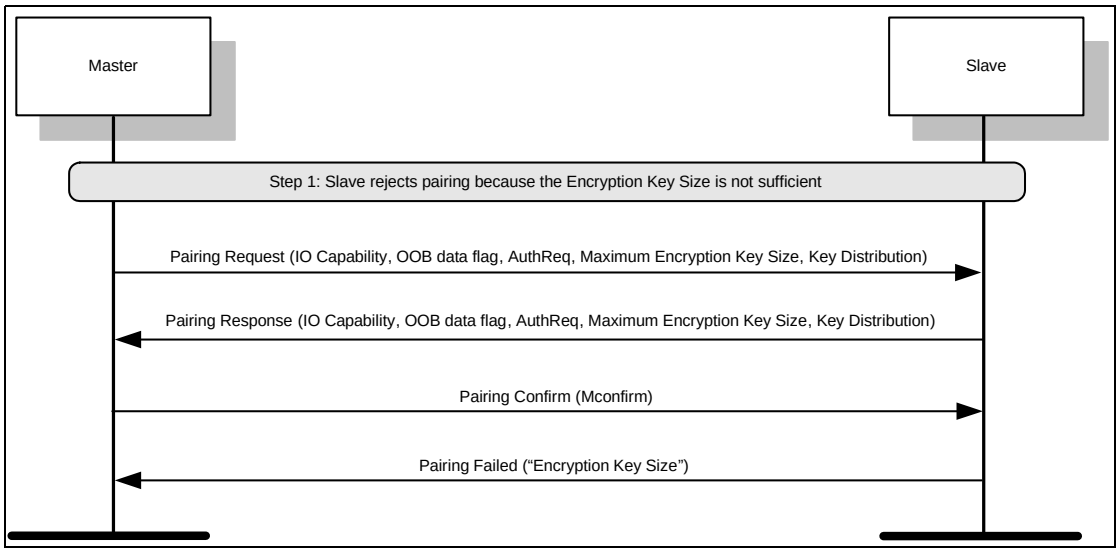



*Figure 5.12: Slave rejects pairing because of key size*

### 5.3.5.4 Passkey Entry Failure on Master

During Passkey Entry pairing the user enters a passkey on both devices. Figure 5.13 shows an example where the passkey entry fails on the master device.

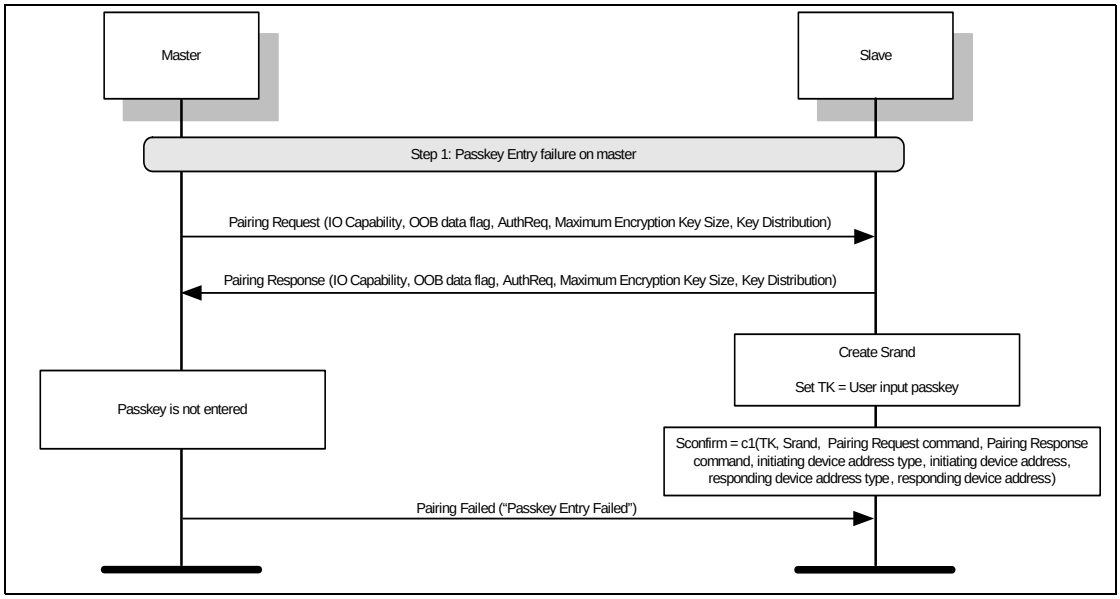



*Figure 5.13: Passkey Entry failure on master*

### *5.3.5.5 **Passkey Entry Failure on Slave***

During Passkey Entry pairing the user enters a passkey on both devices. Figure 5.14 shows an example where the passkey entry fails on the slave device.

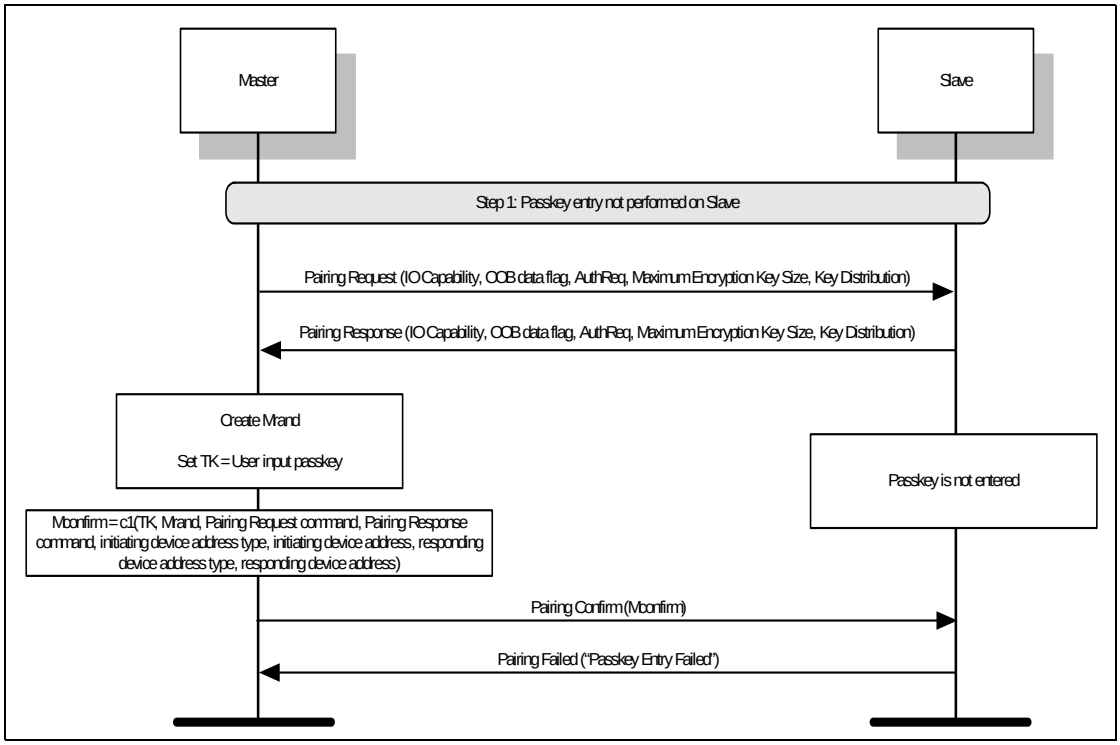



*Figure 5.14: Passkey Entry failure on slave*

### 5.3.5.6 Slave Rejects Master's Confirm Value

During Passkey Entry pairing the user enters a passkey on both devices. Figure 5.15 shows an example where a different passkey is entered on both devices. This sequence could also occur if any of the inputs to c1 (Passkey, Mrand, Srand, Pairing Request command, Pairing Response command, address types or addresses) are incorrect or altered.

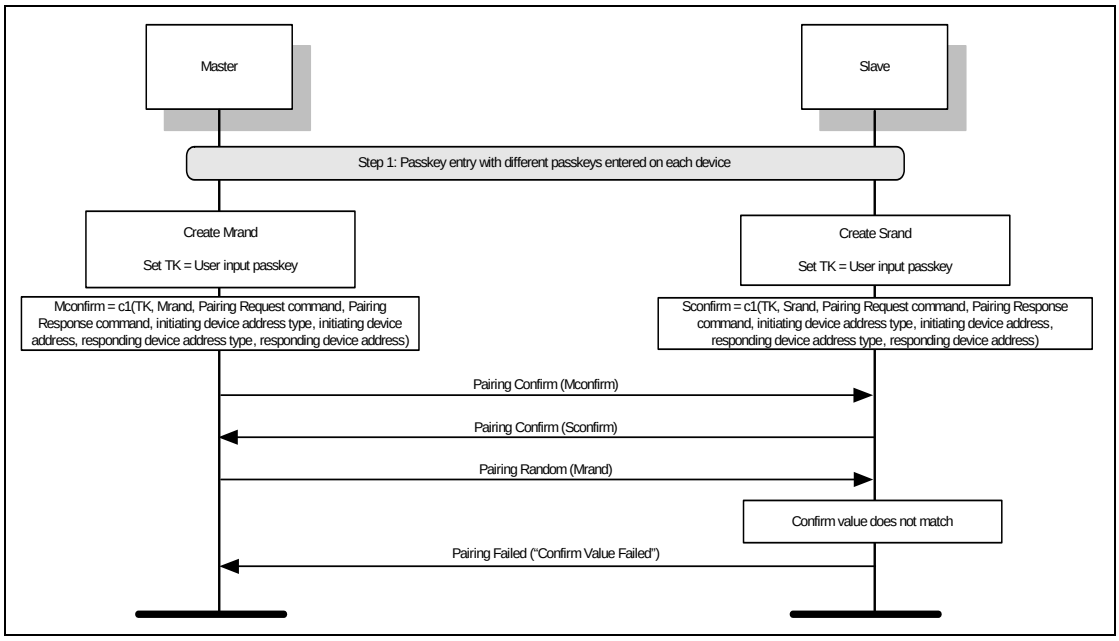



*Figure 5.15: Different Passkeys entered*

### 5.3.5.7 Master Rejects Slaves Confirm Value

During all the pairing methods the master and slave device send random numbers and confirm values. Figure 5.16 shows an example where the master device rejects pairing because it cannot verify the confirm value from the slave because any of the inputs to c1 (Passkey, Mrand, Srand, Pairing Request command, Pairing Response command, address types or addresses) are incorrect or altered.

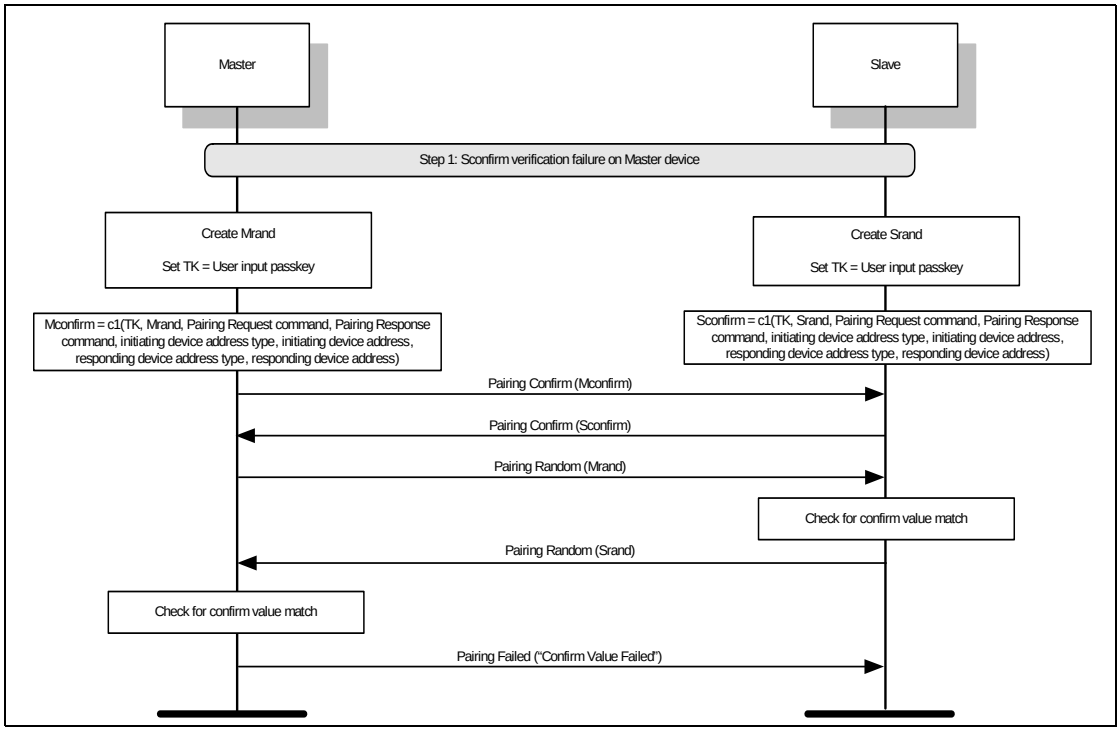


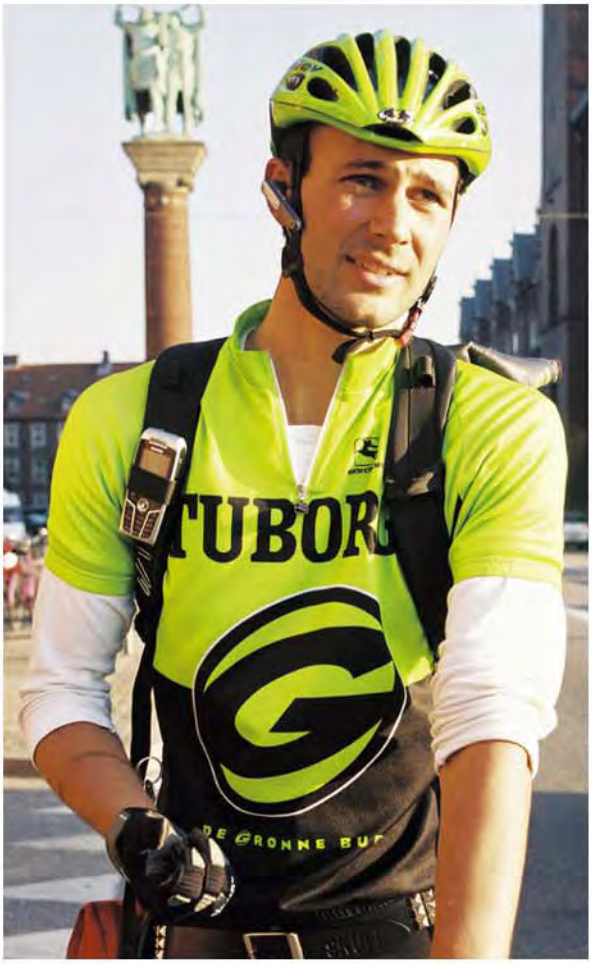

*Figure 5.16: Master rejects Sconfirm value from slave*



Specification Volume 4

# SPECIFICATION OF THE *BLUETOOTH* SYSTEM

Experience More



# Host Controller Interface

## [Transport Layer]

Covered Core Package version:
4.0

Bluetooth®



Ford Ex. 1052

## Revision History

The Revision History is shown in the [Vol. 0, Part C] Appendix on page 55.

## Contributors

The persons who contributed to this specification are listed in the [Vol. 0, Part C] Appendix on page 55.

## Web Site

This specification can also be found on the official Bluetooth web site:
http://www.bluetooth.com

## Disclaimer and Copyright Notice

The copyright in these specifications is owned by the Promoter Members of Bluetooth SIG, Inc. (“Bluetooth SIG”). Use of these specifications and any related intellectual property (collectively, the “Specification”), is governed by the Promoters Membership Agreement among the Promoter Members and Bluetooth SIG (the “Promoters Agreement”), certain membership agreements between Bluetooth SIG and its Adopter and Associate Members (the “Membership Agreements”) and the Bluetooth Specification Early Adopters Agreements (“1.2 Early Adopters Agreements”) among Early Adopter members of the unincorporated Bluetooth special interest group and the Promoter Members (the “Early Adopters Agreement”). Certain rights and obligations of the Promoter Members under the Early Adopters Agreements have been assigned to Bluetooth SIG by the Promoter Members.

Use of the Specification by anyone who is not a member of Bluetooth SIG or a party to an Early Adopters Agreement (each such person or party, a “Member”), is prohibited. The legal rights and obligations of each Member are governed by their applicable Membership Agreement, Early Adopters Agreement or Promoters Agreement. No license, express or implied, by estoppel or otherwise, to any intellectual property rights are granted herein.

Any use of the Specification not in compliance with the terms of the applicable Membership Agreement, Early Adopters Agreement or Promoters Agreement is prohibited and any such prohibited use may result in termination of the applicable Membership Agreement or Early Adopters Agreement and other liability permitted by the applicable agreement or by applicable law to Bluetooth SIG or any of its members for patent, copyright and/or trademark infringement.

THE SPECIFICATION IS PROVIDED “AS IS” WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, SATISFACTORY QUALITY, OR REASONABLE SKILL OR CARE, OR ANY WARRANTY ARISING OUT OF ANY COURSE OF DEALING, USAGE, TRADE PRACTICE, PROPOSAL, SPECIFICATION OR SAMPLE.

Each Member hereby acknowledges that products equipped with the Bluetooth® technology (“Bluetooth® Products”) may be subject to various regulatory controls under the laws and regulations of various governments worldwide. Such laws and regulatory controls may govern, among other things, the combination, operation, use, implementation and distribution of Bluetooth® Products. Examples of such laws and regulatory controls include, but are not limited to, airline regulatory controls, telecommunications regulations, technology transfer controls and health and safety regulations. Each Member is solely responsible for the compliance by their Bluetooth® Products with any such laws and regulations and for obtaining any and all required authorizations, permits, or licenses for their Bluetooth® Products related to such regulations within the applicable jurisdictions. Each Member acknowledges that nothing in the Specification provides any information or assistance in connection with securing such compliance, authorizations or licenses. NOTHING IN THE SPECIFICATION CREATES ANY WARRANTIES, EITHER EXPRESS OR IMPLIED, REGARDING SUCH LAWS OR REGULATIONS.

ALL LIABILITY, INCLUDING LIABILITY FOR INFRINGEMENT OF ANY INTELLECTUAL PROPERTY RIGHTS OR FOR NONCOMPLIANCE WITH LAWS, RELATING TO USE OF THE SPECIFICATION IS EXPRESSLY DISCLAIMED. BY USE OF THE SPECIFICATION, EACH MEMBER EXPRESSLY WAIVES ANY CLAIM AGAINST BLUETOOTH SIG AND ITS PROMOTER MEMBERS RELATED TO USE OF THE SPECIFICATION.

Bluetooth SIG reserves the right to adopt any changes or alterations to the Specification as it deems necessary or appropriate.

Copyright © 1999 - 2010

Ericsson AB,
Lenovo,
Intel Corporation,
Microsoft Corporation,
Motorola, Inc.,
Nokia Corporation,
Toshiba Corporation

*Third-party brands and names are the property of their respective owners.

# CONTENTS

**Part A**
**UART TRANSPORT LAYER**

**Contents ..... 11**
**1 General ..... 13**
**2 Protocol ..... 14**
**3 RS232 Settings ..... 15**
**4 Error Recovery ..... 16**

**Part B**
**USB TRANSPORT LAYER**

**Contents ..... 19**
**1 Overview ..... 21**
**2 USB Endpoint Expectations ..... 25**
2.1 Descriptor Overview ..... 25
2.1.1 Primary Controller Descriptors ..... 25
2.1.2 AMP Controller Descriptors ..... 30
2.2 Control Endpoint Expectations ..... 31
2.2.1 Single FunctionPrimary Controller ..... 31
2.2.2 Primary Controller Function in a Composite Device ..... 31
2.2.3 AMP Controller ..... 32
2.3 Bulk Endpoints Expectations ..... 32
2.4 Interrupt Endpoint Expectations ..... 33
2.5 Isochronous Endpoints Expectations ..... 33
**3 Class Code ..... 34**
3.1 Bluetooth Codes ..... 34
3.1.1 Single Function Primary Controller ..... 34
3.1.2 Single Function AMP Controller ..... 34
3.1.3 Composite Bluetooth Primary and AMP Controller ..... 34
3.1.4 Composite Device Including Bluetooth Primary and AMP Controller ..... 35
**4 Device Firmware Upgrade ..... 36**
**5 Limitations ..... 37**
5.1 Power Specific Limitations ..... 37
5.2 Other Limitations ..... 37
**6 Bluetooth Composite Device Implementation ..... 38**

6.1 Configurations ..... 38
6.2 Using USB Interface Association Descriptors for a Primary Controller Function ..... 38
6.3 Combined Primary Controller Function and Single AMP Controller Function ..... 39

**7 References ..... 43**

**Part C**
**SECURE DIGITAL (SD) TRANSPORT LAYER**

**Contents ..... 47**

**1 Introduction ..... 48**

**2 Goals ..... 49**
2.1 Hardware Goals ..... 49
2.2 Software Goals ..... 49
2.3 Configuration Goals ..... 50
2.4 Configuration for Multiple Controllers ..... 50

**3 Physical Interface Documents ..... 51**

**4 Communication ..... 52**
4.1 Overview ..... 52

**5 Appendix A - Acronyms and Abbreviations ..... 53**

**6 Appendix B - Related Documents ..... 54**

**7 Appendix C - Tests ..... 55**
7.1 Test Suite Structure ..... 55

**Part D**
**THREE-WIRE UART TRANSPORT LAYER**

**Contents ..... 59**

**1 General ..... 61**

**2 Overview ..... 62**

**3 Slip Layer ..... 63**
3.1 Encoding a Packet ..... 63
3.2 Decoding a Packet ..... 63

**4 Packet Header ..... 65**
4.1 Sequence Number ..... 65
4.2 Acknowledge Number ..... 66
4.3 Data Integrity Check Present ..... 66
4.4 Reliable Packet ..... 66
4.5 Packet Type ..... 66

4.6 Payload Length ....67
4.7 Packet Header Checksum ....67
**5 Data Integrity Check ....68**
5.1 16 Bit CCITT-CRC ....68
**6 Reliable Packets ....69**
6.1 Header Checksum Error ....69
6.2 Slip Payload Length Error ....69
6.3 Data Integrity Check Error ....69
6.4 Out Of Sequence Packet Error ....69
6.5 Acknowledgement ....70
6.6 Resending Packets ....70
6.7 Example Reliable Packet Flow ....70
**7 Unreliable Packets ....73**
7.1 Unreliable Packet Header ....73
7.2 Unreliable Packet Error ....73
**8 Link Establishment ....74**
8.1 Uninitialized State ....75
8.2 Initialized State ....75
8.3 Active State ....75
8.4 Sync Message ....76
8.5 Sync Response Message ....76
8.6 Config Message ....76
8.7 Config Response Message ....77
8.8 Configuration Field ....77
8.8.1 Configuration Messages ....78
8.8.2 Sliding Window Size ....78
8.8.3 Level of Data Integrity Check ....78
8.8.4 Out of Frame Software Flow Control ....79
8.8.5 Version Number ....79
**9 LOW POWER ....80**
9.1 Wakeup Message ....80
9.2 Woken Message ....80
9.3 Sleep Message ....81
**10 Out of Frame Control ....82**
10.1 Software Flow Control ....82
**11 Hardware Configuration ....83**
11.1 Wires ....83
11.1.1 Transmit & Receive ....83
11.1.2 Ground ....83
11.2 Hardware Flow ....83

11.2.1 RTS & CTS ..... 83
**12 Recommended Parameters ..... 84**
12.1 Timing Parameters ..... 84
12.1.1 Acknowledgement of Packets ..... 84
12.1.2 Resending Reliable Packets ..... 84
**13 References ..... 85**

# UART TRANSPORT LAYER

***This document describes the UART transport layer (between the Host and the Host Controller). HCI command, event, and data packets flow through this layer, but the layer does not decode them.***


Ford Ex. 1052

# CONTENTS

1 General ..... 13
2 Protocol ..... 14
3 RS232 Settings ..... 15
4 Error Recovery ..... 16

# 1 GENERAL

The objective of this HCI UART Transport Layer is to make it possible to use the Bluetooth HCI over a serial interface between two UARTs on the same PCB. The HCI UART Transport Layer assumes that the UART communication is free from line errors.

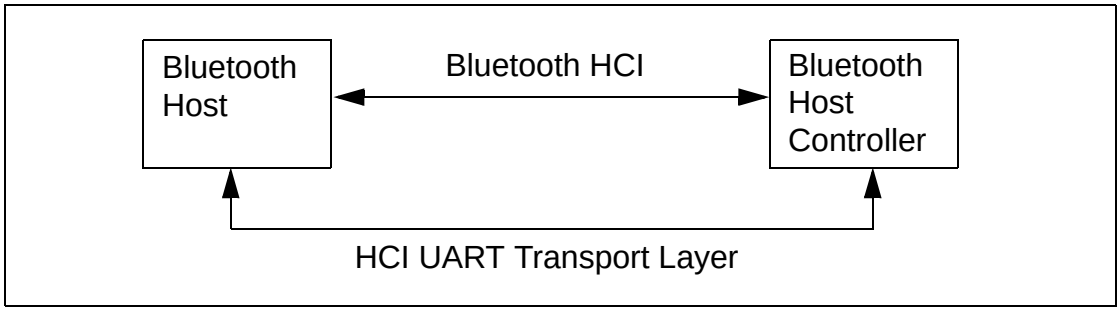



*Figure 1.1: HCI UART Transport Layer*

# 2 PROTOCOL

There are four kinds of HCI packets that can be sent via the UART Transport Layer; i.e. HCI Command Packet, HCI Event Packet, HCI ACL Data Packet and HCI Synchronous Data Packet (see "Host Controller Interface Functional Specification" in Volume 2, Part E). HCI Command Packets can only be sent to the Bluetooth Host Controller, HCI Event Packets can only be sent from the Bluetooth Host Controller, and HCI ACL/Synchronous Data Packets can be sent both to and from the Bluetooth Host Controller.

HCI does not provide the ability to differentiate the four HCI packet types. Therefore, if the HCI packets are sent via a common physical interface, a HCI packet indicator has to be added according to Table 2.1 below.

| HCI packet type | HCI packet indicator |
|---|---|
| HCI Command Packet | 0x01 |
| HCI ACL Data Packet | 0x02 |
| HCI Synchronous Data Packet | 0x03 |
| HCI Event Packet | 0x04 |

*Table 2.1: HCI packet indicators*

The HCI packet indicator shall be sent immediately before the HCI packet. All four kinds of HCI packets have a length field, which is used to determine how many bytes are expected for the HCI packet. When an entire HCI packet has been received, the next HCI packet indicator is expected for the next HCI packet. Over the UART Transport Layer, only HCI packet indicators followed by HCI packets are allowed.

# 3 RS232 SETTINGS

The HCI UART Transport Layer uses the following settings for RS232:

| **Baud rate:** | **manufacturer-specific** |
|---|---|
| Number of data bits: | 8 |
| Parity bit: | no parity |
| Stop bit: | 1 stop bit |
| Flow control: | RTS/CTS |
| Flow-off response time: | manufacturer specific |

*Table 3.1:*

Flow control with RTS/CTS is used to prevent temporary UART buffer overrun. It should not be used for flow control of HCI, since HCI has its own flow control mechanisms for HCI commands, HCI events and HCI data.

If CTS is 1, then the Host/Host Controller is allowed to send.
If CTS is 0, then the Host/Host Controller is not allowed to send.

The flow-off response time defines the maximum time from setting RTS to 0 until the byte flow actually stops.

The RS232 signals should be connected in a null-modem fashion; i.e. the local TXD should be connected to the remote RXD and the local RTS should be connected to the remote CTS and vice versa.

# 4 ERROR RECOVERY

If the Host or the Host Controller lose synchronization in the communication over RS232, then a reset is needed. A loss of synchronization means that an incorrect HCI packet indicator has been detected, or that the length field in an HCI packet is out of range.

If the UART synchronization is lost in the communication from Host to Host Controller, then the Host Controller shall send a Hardware Error Event to tell the Host about the synchronization error. The Host Controller will then expect to receive an HCI_Reset command from the Host in order to perform a reset. The Host Controller will also use the HCI_Reset command in the byte stream from Host to Host Controller to re-synchronize.

If the UART synchronization is lost in the communication from Host Controller to Host, then the Host shall send the HCI_Reset command in order to reset the Host Controller. The Host shall then re-synchronize by looking for the HCI Command Complete event for the HCI_Reset command in the byte stream from Host Controller to Host.

See “Host Controller Interface Functional Specification” for HCI commands and HCI events in the Bluetooth Specification v1.2 or later.

# USB TRANSPORT LAYER

***This document describes the USB transport layer (between a host and the host controller). HCI commands flow through this layer, but the layer does not decode the commands.***


Ford Ex. 1052

# CONTENTS

**1 Overview ..... 21**

**2 USB Endpoint Expectations ..... 25**

2.1 Descriptor Overview ..... 25

2.1.1 Primary Controller Descriptors ..... 25

2.1.2 AMP Controller Descriptors ..... 30

2.2 Control Endpoint Expectations ..... 31

2.2.1 Single FunctionPrimary Controller ..... 31

2.2.2 Primary Controller Function in a Composite Device ..... 31

2.2.3 AMP Controller ..... 32

2.3 Bulk Endpoints Expectations ..... 32

2.4 Interrupt Endpoint Expectations ..... 33

2.5 Isochronous Endpoints Expectations ..... 33

**3 Class Code ..... 34**

3.1 Bluetooth Codes ..... 34

3.1.1 Single Function Primary Controller ..... 34

3.1.2 Single Function AMP Controller ..... 34

3.1.3 Composite Bluetooth Primary and AMP Controller ..... 34

3.1.4 Composite Device Including Bluetooth Primary and AMP Controller ..... 35

**4 Device Firmware Upgrade ..... 36**

**5 Limitations ..... 37**

5.1 Power Specific Limitations ..... 37

5.2 Other Limitations ..... 37

**6 Bluetooth Composite Device Implementation ..... 38**

6.1 Configurations ..... 38

6.2 Using USB Interface Association Descriptors for a Primary Controller Function ..... 38

6.3 Combined Primary Controller Function and Single AMP Controller Function ..... 39

**7 References ..... 43**

# 1 OVERVIEW

This document discusses the requirements of the Universal Serial Bus (USB) interface for Bluetooth hardware. Readers should be familiar with USB, USB design issues, Advanced Configuration Power Interface (ACPI), the overall Bluetooth architecture, and the basics of the radio interface.

The reader should also be familiar with the Bluetooth Host Controller Interface.

Referring to Figure 1.1 through Figure 1.3, notice that this document discusses the implementation details of the two-way arrow labeled “USB Function.”

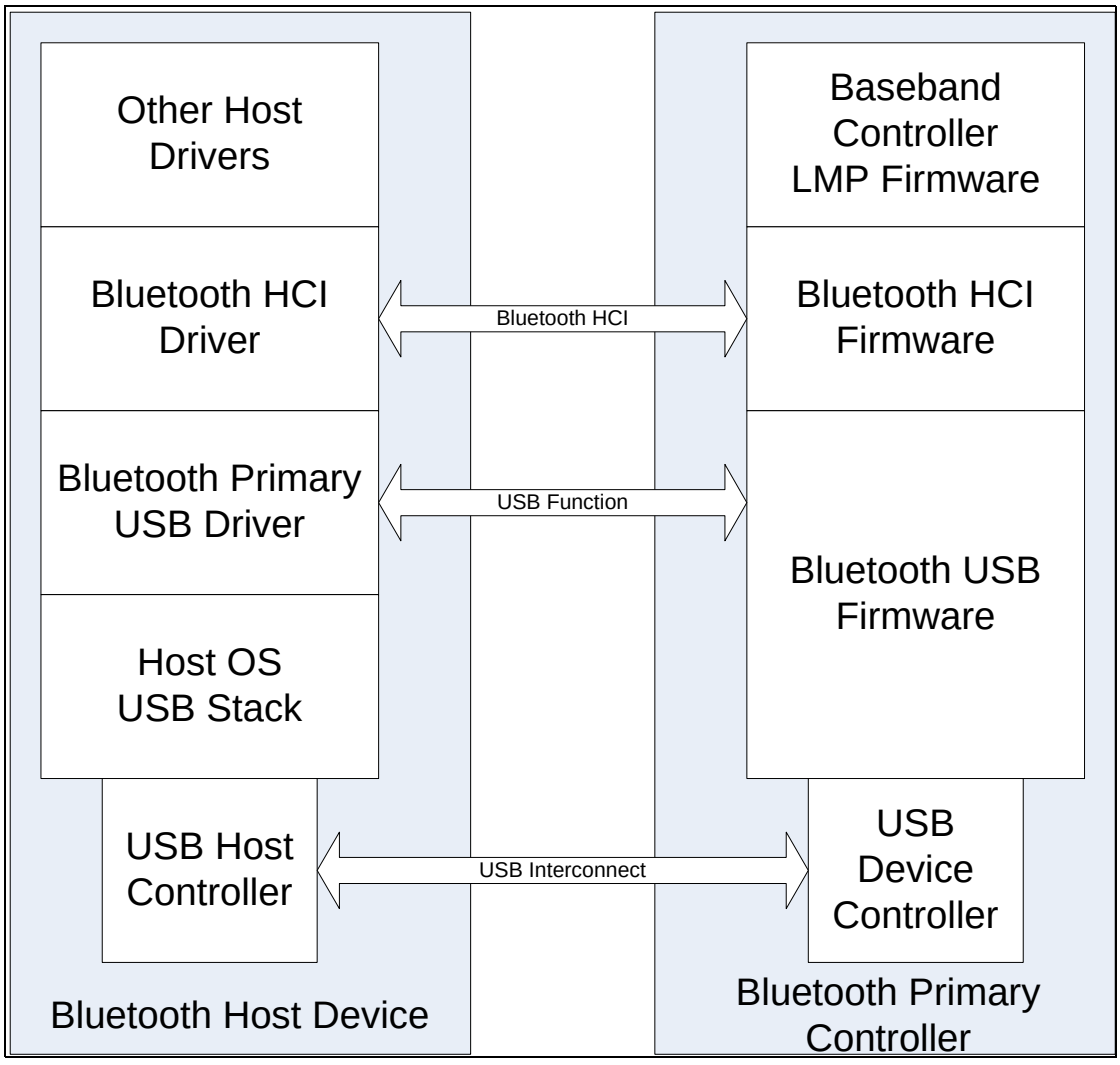



*Figure 1.1: Relationship between the Host and Bluetooth Primary Controller*

This document also specifies implementation of two other configurations:

- USB-connected AMP controller (PAL, MAC, PHY)
- USB-connected composite (multifunction) device that combines a Primary Controller and an AMP Controller.

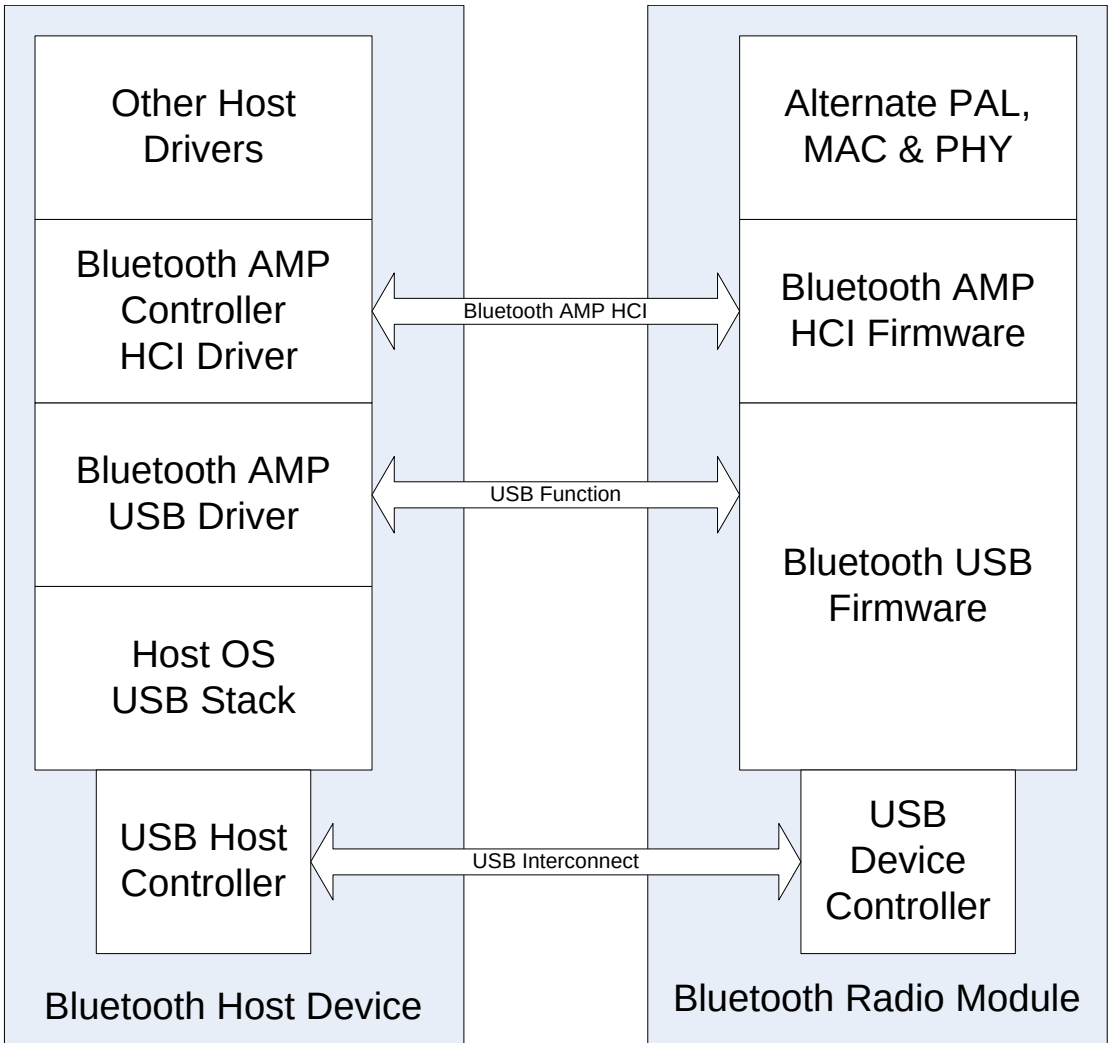



*Figure 1.2: Relationship between the Host and an AMP Controller*

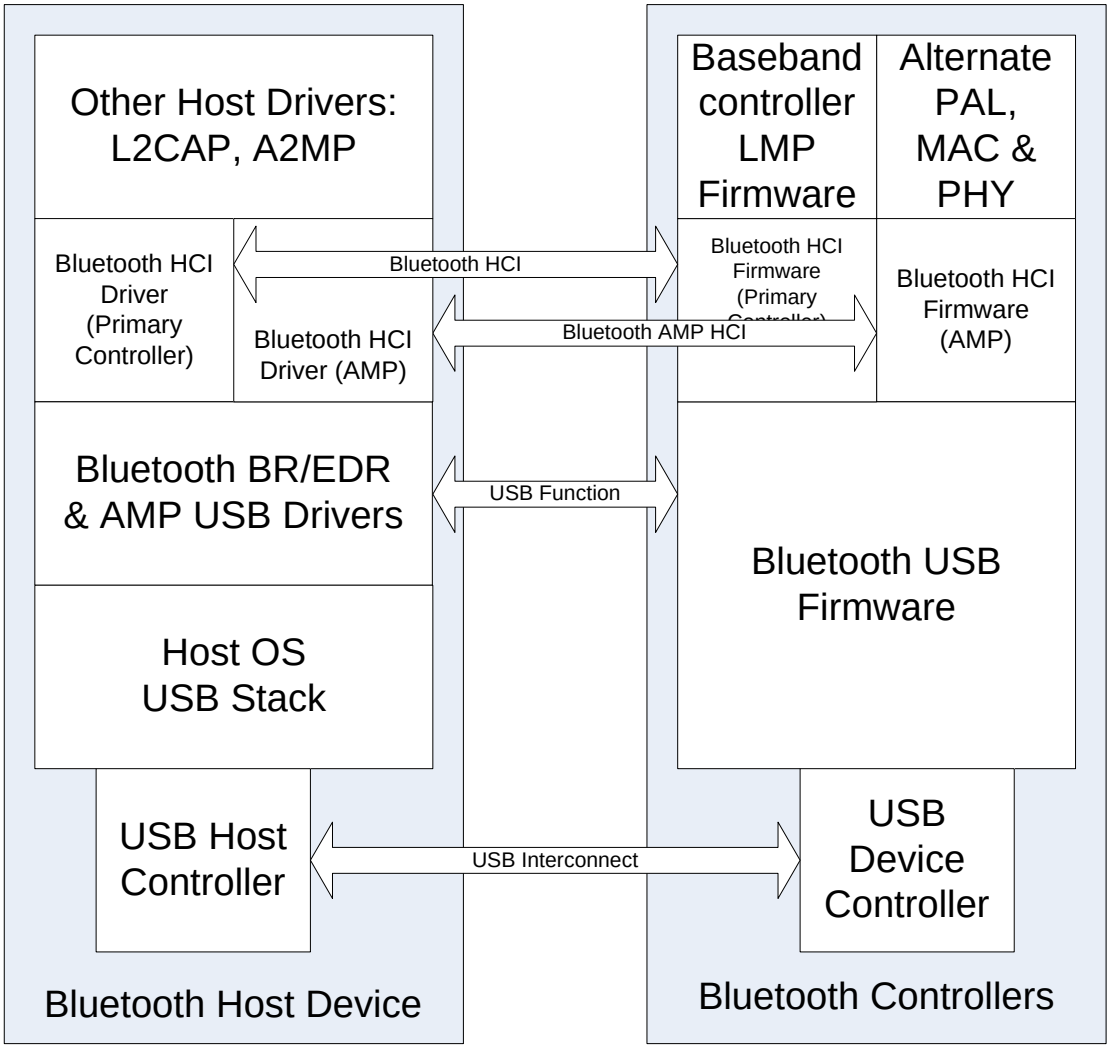



*Figure 1.3: Relationship between Host and a Composite BR/EDR and AMP Controller*

The USB hardware can be embodied in one of several ways:

1. As a USB dongle (e.g. cabled USB)
2. As a USB module integrated into the product and connected internally via a cable or connector
3. Integrated onto the motherboard of a notebook PC or other device and connected via circuit board traces with standard USB, Inter-Chip USB or High Speed Inter-Chip USB
4. Integrated as a subsystem on a single-chip System-on-Chip (SoC) design connected on-chip as part of a compound device.

Finally, for an overview of the connection that is established between two Bluetooth devices, reference Figure 1.4 .

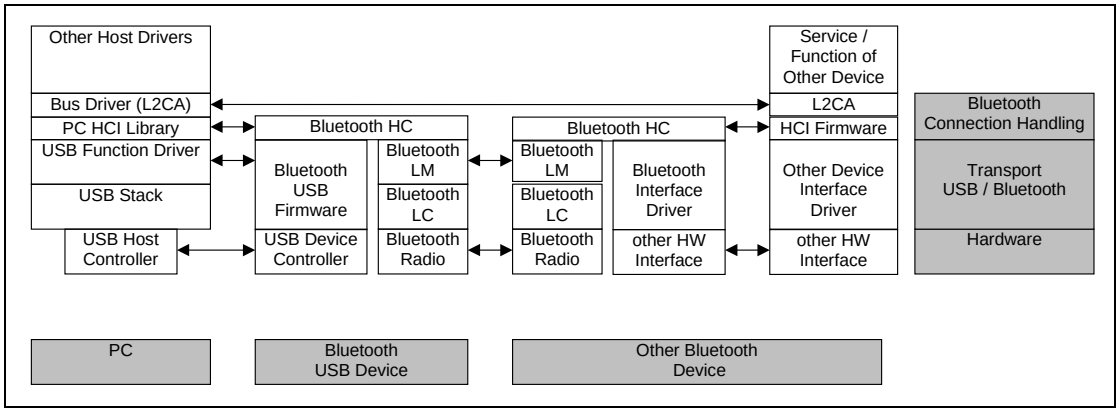



*Figure 1.4: Flow of data from one Bluetooth device to another*

# 2 USB ENDPOINT EXPECTATIONS

This section outlines specific USB endpoints that are required in order to function properly with the host. This section assumes a basic familiarity with USB. The endpoint numbers (labeled 'Suggested Endpoint Address' below) may be dynamically recognized upon driver initialization – this depends on the implementation.

## 2.1 DESCRIPTOR OVERVIEW

The UniversalSerial Bus is intended for high data rates. USB defines several physical layers, ranging from 1.5 Mbps to several Gbps of bus bandwidth. A Bluetooth USB device should provide a USB transport with sufficient bus bandwidth to support the Bluetooth radio transports included in the device.

There are two USB controller types:

- Primary Controller
- AMP Controller

### 2.1.1 Primary Controller Descriptors

The Primary Controller configuration consists of two interfaces. The first interface has no alternate settings and contains the bulk and interrupt endpoints. The second interface provides scalable isochronous bandwidth. The recommended configuration for the second interface has four alternate settings that provide different bandwidth. The default interface is empty so that the device is capable of scaling down to zero isochronous bandwidth.

An HCI packet consisting of an HCI header and HCI data shall be contained in one USB Transfer. A USB transfer is defined by the USB specification as one or more USB transactions that contain the data from one IO request. For example, an ACL data packet containing 256 bytes (both HCI header and HCI data) would be sent over the bulk endpoint in one IO request. That IO request will require four 64-byte full speed USB Transactions or a single 256-byte High-speed USB Transaction, and forms a Transfer. If the Maximum Packet Size for the endpoint on which the transfer is sent is 64 bytes, then that IO request will require four 64-byte USB transactions.

The endpoints are spread across two interfaces so that when adjusting isochronous bandwidth consumption (via select interface calls), any pending bulk and/or interrupt transactions do not have to be terminated or resubmitted.

The following table outlines the recommended configuration.

| Interface Number | Alternate Setting | Suggested Endpoint Address | Endpoint Type | Suggested Max Packet Size |
|---|---|---|---|---|
| **HCI Commands** | | | | |
| N/A | N/A | 0x00 | Control | 8/16/32/64 |
| **HCI Events** | | | | |
| 0 | 0 | 0x81 | Interrupt (IN) | 16 |
| **ACL Data** | | | | |
| 0 | 0 | 0x82 | Bulk (IN) | 32/64 |
| 0 | 0 | 0x02 | Bulk (OUT) | 32/64 |
| **No active voice channels (for USB compliance)** | | | | |
| 1 | 0 | 0x83 | Isoch (IN) | 0 |
| 1 | 0 | 0x03 | Isoch (OUT) | 0 |
| **One voice channel with 8-bit encoding** | | | | |
| 1 | 1 | 0x83 | Isoch (IN) | 9 |
| 1 | 1 | 0x03 | Isoch (OUT) | 9 |
| **Two voice channels with 8-bit encoding or One voice channel with 16-bit encoding** | | | | |
| 1 | 2 | 0x83 | Isoch (IN) | 17 |
| 1 | 2 | 0x03 | Isoch (OUT) | 17 |
| **Three voice channels with 8-bit encoding** | | | | |
| 1 | 3 | 0x83 | Isoch (IN) | 25 |
| 1 | 3 | 0x03 | Isoch (OUT) | 25 |
| **Two voice channels with 16-bit encoding** | | | | |
| 1 | 4 | 0x83 | Isoch (IN) | 33 |
| 1 | 4 | 0x03 | Isoch (OUT) | 33 |
| **Three voice channels with 16-bit encoding** | | | | |
| 1 | 5 | 0x83 | Isoch (IN) | 49 |
| 1 | 5 | 0x03 | Isoch (OUT) | 49 |

*Table 2.1: USB Primary firmware interface and endpoint settings*

The following two examples are used to demonstrate the flow of data given the described endpoints.

| Number of voice channels | Duration of voice data | Encoding |
|---|---|---|
| One | 3 ms per IO Request | 8-bit |

| Time (ms | USB data (header refers to HCI header) (Receive & Send from the host) | Queued data (read / write) | Time (ms) | Air data | Amount Received/ Sent (ms) |
|---|---|---|---|---|---|
| 0 | Receive 0 bytes<br>Send 9 bytes (3 header, 6 data) | 0 / 6 | 0 | Send 0 | 0 / 0 |
| | | 10 / 6 | 0.625 | Receive 10 | 1.25 / 0 |
| 1 | Receive 0 bytes<br>Send 9 bytes (9 bytes HCI data) | 10 / 15 | 1.25 | Send 0 | 1.25 / 0 |
| | | 20 / 15 | 1.875 | Receive 10 | 2.50 / 0 |
| 2 | Receive 0 bytes<br>Send 9 bytes (9 bytes HCI data) | 20 / 24 | 2.50 | Send 0 | 2.50 / 0 |
| | | 30 / 24 | 3.125 | Receive 10 | 3.75 / 0 |
| 3 | Receive 9 bytes (3 header, 6 data)<br>Send 9 bytes (3 header, 6 data) | 24 / 20 | 3.75 | Send 10 | 3.75 / 1.25 |
| 4 | Receive 9 bytes (9 bytes data)<br>Send 9 bytes (9 bytes HCI data) | 25 / 29 | 4.375 | Receive 10 | 5.0 / 1.25 |
| 5 | Receive 9 bytes (9 bytes data)<br>Send 9 bytes (9 bytes HCI data) | 16 / 28 | 5.0 | Send 10 | 5.0 / 2.50 |
| | | 26 / 28 | 5.625 | Receive 10 | 6.25 / 2.50 |
| 6 | Receive 9 bytes (3 header, 6 data)<br>Send 9 bytes (3 header, 6 data) | 20 / 24 | 6.25 | Send 10 | 6.25 / 3.75 |
| | | 30 / 24 | 6.875 | Receive 10 | 7.5 / 3.75 |
| 7 | Receive 9 bytes (9 bytes data)<br>Send 9 bytes (9 bytes HCI data) | 21 / 23 | 7.5 | Send 10 | 7.5 / 5.0 |
| 8 | Receive 9 bytes (9 bytes data)<br>Send 9 bytes (9 bytes HCI data) | 22 / 32 | 8.125 | Receive 10 | 8.75 / 5.0 |
| | | 22 / 22 | 8.75 | Send 10 | 8.75 / 6.25 |
| 9 | Receive 9 bytes (3 header, 6 data)<br>Send 9 bytes (3 header, 6 data) | 26 / 28 | 9.375 | Receive 10 | 10.0 / 6.25 |

*Table 2.2: Example USB single-channel voice traffic data flow*

| Time (ms | USB data (header refers to HCI header) (Receive & Send from the host) | Queued data (read / write) | Time (ms) | Air data | Amount Received/ Sent (ms) |
|---|---|---|---|---|---|
| 10 | Receive 9 bytes (9 bytes data)<br>Send 9 bytes (9 bytes HCI data) | 17 / 27 | 10 | Send 10 | 10.0 / 7.5 |
| | | 27 / 27 | 10.625 | Receive 10 | 11.25 / 7.5 |
| 11 | Receive 9 bytes (9 bytes data)<br>Send 9 bytes (9 bytes HCI data) | 18 / 26 | 11.25 | Send 10 | 11.25 / 8.75 |

*Table 2.2: Example USB single-channel voice traffic data flow*

Convergence is expected because the radio is sending out an average of eight bytes of voice data every one ms and USB is sending eight bytes of voice data every one ms.

| Number of voice channels | Duration of voice data | Encoding |
|---|---|---|
| Two | 3 ms per IO Request | 8-bit |

| Time (ms | USB data (header refers to HCI header) (Receive & Send from the host) | Queued data (read / write) | Time (ms) | Air data | Amount Received / Sent (ms) |
|---|---|---|---|---|---|
| 0 | Receive 0 bytes for Channel #1<br>Send 17 bytes (3 header, 14 data) for Channel #1 | C1- 0/14<br>C2- 0/0 | 0 | Send 0 for C1 | C1- 0/0<br>C2- 0/0 |
| | | C1- 20/ 14<br>C2- 0/0 | 0.625 | Receive 20 for C1 | C1- 2.5/0<br>C2- 0/0 |
| 1 | Receive 0 bytes for Channel #1<br>Send 17 bytes (17 bytes HCI data) for Channel #1 | C1- 20/ 31<br>C2- 0/0 | 1.25 | Send 0 for C2 | C1- 2.5/0<br>C2- 0/0 |
| | | C1- 20/ 31<br>C2- 20/0 | 1.875 | Receive 20 for C2 | C1- 2.5/0<br>C2- 2.5/0 |
| 2 | Receive 0 bytes for Channel #1<br>Send 17 bytes (17 bytes HCI data) for Channel #1 | C1- 20/ 28<br>C2- 20/0 | 2.50 | Send 20 for C1 | C1- 2.5/2.5<br>C2- 2.5/0 |

*Table 2.3: Example USB dual-channel voice traffic data flow*

| Time (ms | USB data (header refers to HCI header) (Receive & Send from the host) | Queued data (read / write) | Time (ms) | Air data | Amount Received / Sent (ms) |
|---|---|---|---|---|---|
| | | C1- 40/ 28<br>C2- 0/0 | 3.125 | Receive 20 for C1 | C1- 5.0/2.5<br>C2- 2.5/0 |
| 3 | Receive 0 bytes for Channel #2<br>Send 17 bytes (3 header, 14 data) for Channel #2 | C1- 40/ 28<br>C2- 20/ 14 | 3.75 | Send 0 for C2 | C1- 5.0/2.5<br>C2- 2.5/0 |
| 4 | Receive 0 bytes for Channel #2<br>Send 17 bytes (17 bytes HCI data) for Channel #2 | C1- 40/ 28<br>C2- 40/ 31 | 4.375 | Receive 20 for C2 | C1- 5.0/2.5<br>C2- 5.0/0 |
| 5 | Receive 0 bytes for Channel #2<br>Send 17 bytes (17 bytes HCI data) for Channel #2 | C1- 40/8<br>C2- 40/ 48 | 5.0 | Send 20 for C1 | C1- 5.0/5.0<br>C2- 5.0/0 |
| | | C1- 60/8<br>C2- 40/ 48 | 5.625 | Receive 20 for C1 | C1- 7.5/5.0<br>C2- 5.0/0 |
| 6 | Receive 17 bytes (3 header, 14 data) for Channel #1<br>Send 17 bytes (3 header, 14 data) for Channel #1 | C1- 46/ 22<br>C2- 40/ 28 | 6.25 | Send 20 for C2 | C1- 7.5/5.0<br>C2- 5.0/2.5 |
| | | C1- 46/ 22<br>C2- 60/ 28 | 6.875 | Receive 20 for C2 | C1- 7.5/5.0<br>C2- 7.5/2.5 |
| 7 | Receive 17 bytes (17 bytes data) for Channel #1<br>Send 17 bytes (17 bytes HCI data) for Channel #1 | C1- 29/ 19<br>C2- 60/ 28 | 7.5 | Send 20 for C1 | C1- 7.5/7.5<br>C2- 7.5/2.5 |
| 8 | Receive 17 bytes (17 bytes data) for Channel #1<br>Send 17 bytes (17 bytes HCI data) for Channel #1 | C1- 32/ 36<br>C2- 60/ 28 | 8.125 | Receive 20 for C1 | C1- 10/7.5<br>C2- 7.5/2.5 |
| | | C1- 32/ 36<br>C2- 60/8 | 8.75 | Send 20 for C2 | C1- 10/7.5<br>C2- 7.5/5.0 |

*Table 2.3: Example USB dual-channel voice traffic data flow*

| Time (ms | USB data (header refers to HCI header) (Receive & Send from the host) | Queued data (read / write) | Time (ms) | Air data | Amount Received / Sent (ms) |
|---|---|---|---|---|---|
| 9 | Receive 17 bytes (3 header, 14 data) for Channel #2<br>Send 17 bytes (3 header, 14 data) for Channel #2 | C1- 32/ 36<br>C2- 54/ 22 | 9.375 | Receive 20 for C2 | C1- 10/7.5<br>C2- 10/5.0 |
| 10 | Receive 17 bytes (17 bytes data) for Channel #2<br>Send 17 bytes (17 bytes HCI data) for Channel #2 | C1- 32/ 16<br>C2- 37/ 39 | 10 | Send 20 for C1 | C1- 10/10<br>C2- 10/5.0 |
| | | C1- 52/ 16<br>C2- 37/ 39 | 10.625 | Receive 20 for C1 | C1- 12.5/ 10<br>C2- 10/5.0 |
| 11 | Receive 17 bytes (17 bytes data) for Channel #2<br>Send 17 bytes (17 bytes HCI data) for Channel #2 | C1- 52/ 16<br>C2- 20/ 36 | 11.25 | Send 20 for C2 | C1- 12.5/ 10<br>C2- 10/7.5 |

*Table 2.3: Example USB dual-channel voice traffic data flow*

### 2.1.2 AMP Controller Descriptors

The AMP Controller configuration consists of one interface. That interface (interface N - see device descriptors) has no alternate settings and contains the bulk and interrupt endpoints.

An HCI packet, consisting of an HCI header and HCI data, shall be contained in one USB transfer.

| Interface Number | Alternate Setting | Suggested Endpoint Address | Endpoint Type | Suggested Max Packet Size |
|---|---|---|---|---|
| **HCI Commands** | | | | |
| N/A | N/A | 0x00 | Control | 64 |
| **HCI Events** | | | | |
| N | 0 | 0x81 | Interrupt (IN) | 512 |
| **AMP Data** | | | | |
| N | 0 | 0x82 | Bulk (IN) | 512 |
| N | 0 | 0x02 | Bulk (OUT) | 512 |

*Table 2.4: USB AMP firmware interface and endpoint settings*

NOTE: Endpoint addresses are reported by the device in the Endpoint Descriptors.

## 2.2 CONTROL ENDPOINT EXPECTATIONS

Endpoint 0 is used to configure and control the USB device. Endpoint 0 will also be used to allow the host to send HCI-specific commands to the host controller.

All HCI Control Packets delivered to Endpoint 0 are addressed in the Setup Data structure (See 9.3 of [2]). This structure contains fields which determine the destination within the device. The bmRequestType can be used to select the Device or the Interface. If Interface is selected, the wIndex parameter must select the Index for the targeted Bluetooth controller.

### 2.2.1 Single FunctionPrimary Controller

For a single function Primary Controller, the Host should address HCI command packets to the Device. HCI command packets should be sent with the following parameters:

| Parameter | Value | description |
|---|---|---|
| bmRequestType | 0x20 | Host-to-device class request, device as target |
| bRequest | 0x00 | |
| wValue | 0x00 | |
| wIndex | 0x00 | |

*Table 2.5: HCI command packet addressing for single-function Primary controllers*

Note: For historical reasons, if the Primary Controller firmware receives a packet over this endpoint, it should treat the packet as an HCI command packet regardless of the value of bRequest, wValue and wIndex. Some Host devices set bRequest to 0xE0.

### 2.2.2 Primary Controller Function in a Composite Device

For a Primary Controller included in a composite (multi-function) device, the host should address HCI control packets to the Interface of the Primary Controller. HCI command packets should be sent with the following parameters:

| Parameter | Value | description |
|---|---|---|
| bmRequestType | 0x21 | Host-to-Interface class request, interface as target |
| bRequest | 0x00 | |
| wValue | 0x00 | |

*Table 2.6: HCI command packet addressing for composite device Primary Controllers*

| Parameter | Value | description |
|---|---|---|
| wIndex | IF# | This is the actual Interface number within the composite device. |

*Table 2.6: HCI command packet addressing for composite device Primary Controllers*

If the Host system driver addresses USB requests containing HCI command packets to the Device (see Section 2.2.1) instead of to the Interface, the device implementation shall recognize these HCI command packets and correctly route them to the Primary Controller function, to ensure correct operation of the Primary Controller function and avoid malfunctions in other functions contained in the composite device.

### 2.2.3 AMP Controller

For any AMP Controller, either in a single function device or included in a composite device, the Host shall address HCI control packets to the Interface of the AMP Controller. HCI command packets shall be sent with the following parameters:

| Parameter | Value | description |
|---|---|---|
| bmRequestType | 0x21 | Host-to-Interface class request, interface as target |
| bRequest | 0x2B | Arbitrary value chosen to identify requests targeted to an AMP controller function. |
| wValue | 0x00 | |
| wIndex | IF# | This is the actual Interface number within the composite device. |

*Figure 2.1: HCI command packet addressing for single-function AMP Controller*

## 2.3 BULK ENDPOINTS EXPECTATIONS

Data integrity is a critical aspect for ACL data. This, in combination with bandwidth requirements, is the reason for using a bulk endpoint. Multiple 64-byte packets can be shipped per USB Frame (1 millisecond, full speed) or 512-byte packets per USB Microframe (125 microseconds, high-speed), across the bus.

Suggested bulk max packet size is 64 bytes for full-speed, or 512 bytes for high speed.

Bulk has the ability to detect errors and correct them. In order to avoid starvation, a flow control model similar to the shared endpoint model is recommended for the host controller.

## 2.4 INTERRUPT ENDPOINT EXPECTATIONS

An interrupt endpoint is necessary to ensure that events are delivered in a predictable and timely manner. Event packets can be sent across USB with a guaranteed latency.

The interrupt endpoint should have an interval of 1 ms (full speed). For a controller using USB high-speed the interrupt interval may have an interval of 125 microseconds.

The USB software and firmware requires no intimate knowledge of the events passed to the host controller.

## 2.5 ISOCHRONOUS ENDPOINTS EXPECTATIONS

These isochronous endpoints transfer synchronous data to and from the host controller of the radio.

Time is the critical aspect for this type of data. The USB firmware should transfer the contents of the data to the host controllers' synchronous FIFOs. If the FIFOs are full, the data should be overwritten with new data.

These endpoints have a one (1) ms interval, as required by Chapter 9 of the USB Specification, Versions 1.0 and 1.1.

The radio is capable of three (3) 64Kb/s voice channels (and can receive the data coded in different ways – 16-bit linear audio coding is the method that requires the most data). A suggested max packet size for this endpoint would be at least 64 bytes. (It is recommended that max packet sizes be on power of 2 boundaries for optimum throughput.) However, if it is not necessary to support three voice channels with 16-bit coding, 32 bytes could also be considered an acceptable max packet size.

# 3 CLASS CODE

A class code will be used that is specific to all USB Bluetooth devices. This will allow the proper driver stack to load, regardless of which vendor built the device.

## 3.1 BLUETOOTH CODES

The following values are defined for Bluetooth Devices:

| Code | Label | Value | Description |
|---|---|---|---|
| Class | bDeviceClass | 0xE0 | Wireless Controller |
| Subclass | bDeviceSubClass | 0x01 | RF Controller |
| Protocol | bDeviceProtocol | 0x01 | Bluetooth Primary Controller |

*Table 3.1: USB Codes for Primary Controllers*

| Code | Label | Value | Description |
|---|---|---|---|
| Class | bDeviceClass | 0xE0 | Wireless Controller |
| Subclass | bDeviceSubClass | 0x01 | RF Controller |
| Protocol | bDeviceProtocol | 0x04 | Bluetooth AMP controller |

*Table 3.2: USB Codes for AMP Controllers*

These values should also be used in the interface descriptors for the interfaces described in Section 2.1 and as described in the following sections.

### 3.1.1 Single Function Primary Controller

Set the Class, Subclass, and Protocol values for the Device Descriptor and for each Interface Descriptor which applies to the Primary Controller as defined in Table 3.1.

### 3.1.2 Single Function AMP Controller

Set the Class, Subclass, and Protocol values for the Device Descriptor and for each Interface Descriptor which applies to the AMP Controller as defined in Table 3.2.

### 3.1.3 Composite Bluetooth Primary and AMP Controller

Set the Class, Subclass, and Protocol values for the Device Descriptor as defined in the USB Interface Association Descriptor (IAD) ECN.

Set the Class, Subclass, and Protocol values for each Interface Descriptor which applies to the Primary Controller as defined in Table 3.1.

| Code | Label | Value | Description |
|---|---|---|---|
| Class | bDeviceClass | 0xEF | Miscellaneous |
| Subclass | bDeviceSubClass | 0x02 | Common Class |
| Protocol | bDeviceProtocol | 0x01 | Interface Association Descriptor |

*Table 3.3: USB Codes for composite devices using IAD*

Set the Class, Subclass, and Protocol values for the each Interface Descriptor which applies to the AMP Controller as defined in Table 3.2.

### 3.1.4 Composite Device Including Bluetooth Primary and AMP Controller

Set the Class, Subclass, and Protocol values for the Device Descriptor to as defined in Table 3.3.

Set the Class, Subclass, and Protocol values for each Interface Descriptor which applies to the Primary Controller as defined in Table 3.1.

Set the Class, Subclass, and Protocol values for the each Interface Descriptor which applies to the AMP Controller as defined in Table 3.2.

# 4 DEVICE FIRMWARE UPGRADE

Firmware upgrade capability is not a required feature. But if implemented, the firmware upgrade shall be compliant with the “Universal Serial Bus Device Class Specification for Device Firmware Upgrade” (version 1.0 dated May 13, 1999) available on the USB Forum web site at http://www.usb.org.

# 5 LIMITATIONS

## 5.1 POWER SPECIFIC LIMITATIONS

Some USB host controllers in portable devices will not receive power while the system is in a sleep mode. For example, many PCs do not supply power to the USB port in system power states S3 or S4, as defined in ACPI. Hence, USB wake-up can only occur when the system is in S1 or S2. Furthermore, all connections and state information of the USB Bluetooth controller will be lost in the system sleep state if power is lost necessitating re-initialization when the device returns to the active state.

Some USB host controllers further continually snoop memory when a device is attached to see if there is any work that needs to be done. The snoop is typically performed every 1ms for USB full-speed devices. This prevents the processor from dropping into a low power state known as C3. Because the processor is not able to enter the C3 state, significant power consumption may occur. This is a major concern for battery-powered hosts such notebook computers. Some host controllers are capable of scheduling polling of USB devices at short intervals while snooping the host's memory much less frequently. Systems with such host controllers may be able to greatly increase the percentage of time spent in the C3 state even if Bluetooth connections are maintained.

A feature called Link Power Management is also recommended for implementation by Bluetooth devices. It is described in an ECN (Engineering Change Notice) from the USB Implementers' Forum.

## 5.2 OTHER LIMITATIONS

Data corruption may occur across isochronous endpoints. Endpoints one and two may suffer from data corruption.

USB provides 16-CRC on all data transfers. The USB has a bit error rate of $10^{-13}$.

*Note that when a dongle is removed from the system, the radio will lose power (assuming this is a bus-powered device). This means that devices will lose connection.*

# 6 BLUETOOTH COMPOSITE DEVICE IMPLEMENTATION

A USB Composite contains multiple independent functions. This section describes how to implement Bluetooth functions within a USB Composite device. This may require the use of Interface Association Descriptors (IAD) to aggregate multiple Interfaces. This also requires the host to address USB requests to the specific Interface (see [1]).

## 6.1 CONFIGURATIONS

There are several ways that Bluetooth Controller functions may be included in a USB composite device:

- Primary Controller and AMP Controller (see Section 6.3)
- Primary Controller in a multi-radio device
- Primary Controller in a device also containing non-radio functions (e.g. memory)
- AMP Controller in a multi-radio device

## 6.2 USING USB INTERFACE ASSOCIATION DESCRIPTORS FOR A Primary CONTROLLER FUNCTION

A Primary Controller (ref) shall contain at least two interfaces:

- HCI Events and ACL data (3 endpoints)
- HCI SCO data (2 endpoints, multiple alternate settings)

A Primary Controller may also contain

- Device Firmware Upgrade (see [2])

When used in a USB Composite device, a Primary Controller function shall use an IAD descriptor to associate the provided interfaces. The following is an example IAD for a Primary function without Device Firmware Upgrade:

- It would be contained within a Configuration Descriptor set.
- It would be followed by two Interface Descriptors and associated Endpoint Descriptors.

| Offset | Field | Size | Value | Description |
|---|---|---|---|---|
| 0 | bLength | 1 | 0x08 | Size of this descriptor in octets |
| 1 | bDescriptorType | 1 | 0x0B | INTERFACE ASSOCIATION DESCRIPTOR |

*Table 6.1: Example Interface Association Descriptor used for a Primary Controller function*

| Offset | Field | Size | Value | Description |
|---|---|---|---|---|
| 2 | bFirstInterface | 1 | number | Interface number of the first interface associated with this device |
| 3 | bInterfaceCount | 1 | 0x02 | Number of contiguous interfaces associated with the function |
| 4 | bFunctionClass | 1 | 0xE0 | Wireless Controller |
| 5 | bFunctionSubClass | 1 | 0x01 | RF Controller |
| 6 | bFunctionProtocol | 1 | 0x01 | Bluetooth Primary Controller |
| 7 | iFunction | 1 | Index | Pointer to a name string for this function, if any is provide |

*Table 6.1: Example Interface Association Descriptor used for a Primary Controller function*

## 6.3 COMBINED PRIMARY CONTROLLER FUNCTION AND SINGLE AMP CONTROLLER FUNCTION

An AMP Controller (see Volume 5) shall contain at least one interface:

- HCI Events and ACL Data (3 endpoints)

An AMP Controller may also contain

- Device Firmware Upgrade (see [2])

When used in a USB Composite device, an AMP Controller does not need IAD. If the device contains the Device Firmware Upgrade option as a separate Device function, an IAD is not needed. If the Device Firmware Upgrade option is bundled with the AMP Controller function, then an IAD is needed to bind the two interfaces.

A USB Composite device containing only a Primary Controller and an AMP Controller may include a Configuration Descriptor set with the following structure:

| Offset | Field | Size | Value | Description |
|---|---|---|---|---|
| | | | | Configuration Descriptor |
| 0 | bLength | 1 | 0x09 | Configuration Descriptor size |
| 1 | bDescriptorType | 1 | 0x02 | CONFIGURATION DESCRIPTOR |
| 2 | wTotalLength | 2 | | Size of the entire bundle |

*Table 6.2: Example Configuration Descriptor used for a composite Primary and AMP Controller device*

| Offset | Field | Size | Value | Description |
|---|---|---|---|---|
| 4 | bNumInterfaces | 1 | 0x03 | Three Interfaces:<br>IF#1 Primary events & ACL<br>IF#2 Primary SCO or eSCO<br>IF#3 AMP events & ACL |
| 5 | bConfigurationValue | 1 | 0x01 | |
| 6 | iConfiguration | 1 | Index | Reference to a string describing this. |
| 7 | bmAttributes | 1 | | Attributes bitmap |
| 8 | MaxPower | 1 | 0xFA | Max power used/2ma, e.g. 500ma |
| | | | | Interface Association Descriptor |
| 0 | bLength | 1 | 0x08 | Interface Association Descriptor size |
| 1 | bDescriptorType | 1 | 0x0B | INTERFACE ASSOCIATION DESCRIPTOR |
| 2 | bFirstInterface | 1 | IF#1 | Interface number of the first interface associated with this device |
| 3 | bInterfaceCount | 1 | 0x03 | Number of contiguous interfaces associated with the function. |
| 4 | bFunctionClass | 1 | 0xE0 | Wireless Controller |
| 5 | bFunctionSubClass | 1 | 0x01 | RF Controller |
| 6 | bFunctionProtocol | 1 | 0x01 | Bluetooth Primary Controller |
| 7 | iFunction | 1 | Index | Pointer to a name string for this function, if any is provided. |
| | | | | Primary Events & ACL Interface |
| 0 | bLength | 1 | 0x09 | |
| 1 | bDescriptorType | 1 | 0x04 | INTERFACE DESCRIPTOR |
| 2 | bInterfaceNumber | 1 | IF#1 | This is the first interface in the device. |
| 3 | bAlternateSetting | 1 | 0x00 | This is the first setting for this interface. |
| 4 | bNumEndpoints | 1 | 0x03 | INT, Bulk OUT (ACL), Bulk IN (ACL) |
| 5 | bFunctionClass | 1 | 0xE0 | Wireless Controller |

*Table 6.2: Example Configuration Descriptor used for a composite Primary and AMP Controller device*

| Offset | Field | Size | Value | Description |
|---|---|---|---|---|
| 6 | bFunctionSubClass | 1 | 0x01 | RF Controller |
| 7 | bFunctionProtocol | 1 | 0x01 | Bluetooth Primary Controller |
| 8 | iInterface | 1 | Index | Pointer to a name string for this interface, if any is provided. |
| | Omitted | | | INT ENDPOINT DESCRIPTOR |
| | Omitted | | | BULK OUT ENPOINT DESCIP-TOR |
| | Omitted | | | BULK IN ENPOINT DESCIP-TOR |
| | | | | Primary Controller SCO Inter-face |
| 0 | bLength | 1 | 0x09 | |
| 1 | bDescriptorType | 1 | 0x04 | INTERFACE DESCRIPTOR |
| 2 | bInterfaceNumber | 1 | IF#2 | This is the second interface in the device. |
| 3 | bAlternateSetting | 1 | 0x00 | This is the first setting for this interface |
| 4 | bNumEndpoints | 1 | 0x02 | Isoch OUT (SCO), Isoch IN (SCO) |
| 5 | bFunctionClass | 1 | 0xE0 | Wireless Controller |
| 6 | bFunctionSubClass | 1 | 0x01 | RF Controller |
| 7 | bFunctionProtocol | 1 | 0x01 | Bluetooth Primary Controller |
| 8 | iInterface | 1 | Index | Pointer to a name string for this interface, if any is provided. |
| | Omitted | | | ISOCH OUT ENDPOINT DESCRIPTOR |
| | Omitted | | | ISOCH IN ENPOINT DESCRIP-TOR |
| | | | | AMP Controller: Events & ACL Interface |
| 0 | bLength | 1 | 0x09 | |
| 1 | bDescriptorType | 1 | 0x04 | INTERFACE DESCRIPTOR |
| 2 | bInterfaceNumber | 1 | IF#3 | This is the third interface in the device. |
| 3 | bAlternateSetting | 1 | 0x00 | This is the first setting for this interface. |

*Table 6.2: Example Configuration Descriptor used for a composite Primary and AMP Controller device*

| Offset | Field | Size | Value | Description |
|---|---|---|---|---|
| 4 | bNumEndpoints | 1 | 0x03 | INT, Bulk OUT (ACL), Bulk IN (ACL) |
| 5 | bFunctionClass | 1 | 0xE0 | Wireless Controller |
| 6 | bFunctionSubClass | 1 | 0x01 | RF Controller |
| 7 | bFunctionProtocol | 1 | 0x04 | Bluetooth AMP Controller |
| 8 | iInterface | 1 | Index | Pointer to a name string for this interface, if any is provided. |
| | Omitted | | | INT ENDPOINT DESCRIPTOR |
| | Omitted | | | BULK OUT ENPOINT DESCIP-TOR |
| | Omitted | | | BULK IN ENPOINT DESCIP-TOR |

*Table 6.2: Example Configuration Descriptor used for a composite Primary and AMP Controller device*

# 7 REFERENCES

[1] USB 2.0, http://www.usb.org/developers/docs/docs, including:
1. Interface Association Descriptors ECN
2. Inter-Chip USB Supplement 1.0
3. High Speed Inter-Chip USB Supplement 1.0
4. Link Power Management 1.0

[2] USB Device Firmware Upgrade 1.1

# SECURE DIGITAL (SD) TRANSPORT LAYER

***This document describes the SD transport layer (between the Host and Controller). HCI command, event and data packets flow through this layer, but the layer does not decode them. The Bluetooth SD Transport layer is defined in a document owned and maintained by the Secure Digital Association. Information regarding that document is described herein.***


Ford Ex. 1052

# CONTENTS

**1 Introduction ..... 48**

**2 Goals ..... 49**

2.1 Hardware Goals ..... 49

2.2 Software Goals ..... 49

2.3 Configuration Goals ..... 50

2.4 Configuration for Multiple Controllers ..... 50

**3 Physical Interface Documents ..... 51**

**4 Communication ..... 52**

4.1 Overview ..... 52

**5 Appendix A - Acronyms and Abbreviations ..... 53**

**6 Appendix B - Related Documents ..... 54**

**7 Appendix C - Tests ..... 55**

7.1 Test Suite Structure ..... 55

# 1 INTRODUCTION

This document discusses the requirements of the Secure Digital (SD) interface for Bluetooth hardware. Readers should be familiar with SD, SD design issues, and the overall Bluetooth architecture. The reader should also be familiar with the Bluetooth Host Controller Interface.

The SD Bluetooth Protocol is documented in the SDIO Card Type-A Specification for Bluetooth, which is owned and maintained by the Secure Digital Association (SDA). The full specification is available to members of the SDA that have signed all appropriate SD NDA and license requirements. The SDA also makes a Non-NDA version available, the Simplified Version of: SDIO Card Type-A Specification for Bluetooth. There are no changes to the SDA document to comply with the requirements of the Bluetooth SIG.

# 2 GOALS

## 2.1 HARDWARE GOALS

The Bluetooth SD transport interface specification is designed to take advantage of both the SD Physical Transport bus and the packet orientation of the Bluetooth HCI protocol. Thus, all data is transferred in blocks as packets. Since the block size used on the SD bus may be smaller than the HCI packet, a segmentation and recombination protocol is defined.

SDIO [2] provides different data rate options, including different bit path widths and clock rates. Systems using SDIO should choose options that provide sufficient bandwidth to support the needs of the Controller, both for device control (HCI commands and events) and for data (ACL, SCO).

The specification supports SDIO-connected Controller:

- Primary Controller
- AMP Controller
- Multifunction Controller incorporating some combination of Primary and AMP Controllers.

## 2.2 SOFTWARE GOALS

The Bluetooth SD transport interface specification is designed for non-embedded solutions. It is assumed that the host software does not necessarily have a priori knowledge of the SD Bluetooth device.

The interface is not designed for embedded applications where much of the information passed via the interface is known in advance.

The SDA also defines a Bluetooth interface for embedded applications where the Controller contains protocol layers above HCI (RFComm, SDP etc.). This specification is called SDIO Card Type-B Specification for Bluetooth. Information about this specification can be obtained from the SDA:

http://www.sdcard.org.

## 2.3 CONFIGURATION GOALS

The SDIO Card Specification [2] defines SDIO Standard Function Codes in Table 6-4:

0x2 This function supports the SDIO Type-A for Bluetooth standard interface

0x3 This function supports the SDIO Type-B for Bluetooth standard interface

0x9 This function supports the SDIO Type-A for Bluetooth AMP standard interface

The SDIO Card Type-A Specification for Bluetooth [3] specifies how to implement a Primary Controller. Table 2.1 defines Service ID codes to route HCI messages (codes 0x01 - 0x04). An AMP Controller shall conform to [3] except for one Service ID code:

| SDIO Type-A service ID | Primary Controller | AMP Controller |
|---|---|---|
| 0x00 | reserved | reserved |
| 0x01 | HCI Command Packet | HCI Command Packet |
| 0x02 | ACL Data | ACL Data: Best Effort and Guaranteed Logical Links |
| 0x03 | SCO Data | reserved |
| 0x04 | HCI Event Packet | HCI Event Packet |
| 0x05-0xFF | Reserved as per [3] | Reserved as per [3] |

*Table 2.1: Bluetooth SDIO Controller Service ID codes*

## 2.4 CONFIGURATION FOR MULTIPLE CONTROLLERS

An SDIO device may contain one or more Controllers as defined in [3]. These Controller functions shall conform to the requirements of [2] section 6.12.

# 3 PHYSICAL INTERFACE DOCUMENTS

This specification references the SD SDIO Card Type-A Specification for Bluetooth. This SDA document defines the Bluetooth HCI for all SD devices that support an HCI level interface. Any SD Bluetooth device claiming compliance with the SD Bluetooth Transport must support this interface and additionally adhere to its device type specification, which is set by the Secure Digital Association. The SDIO Card Type-A Specification for Bluetooth document is based on the SDIO Card Specification, which in turn is based on the SD Memory Card Specification: Part 1 Physical Layer Specification. All of these documents are copyrighted by the SDA and are available ONLY to SDA member companies that have signed the appropriate NDA documents with the SDA. As an introduction to the SD Bluetooth Type A specification, the SDA has created 'Simplified' versions of each of these documents. The simplified versions do not contain enough information to fully implement a device, however they do contain enough information to convey the structure and intent of the specifications.

Applicable SDA Documents available to members of the SDA:

**SD Memory Card Specification: Part 1 Physical Layer Specification**

**SDIO Card Specification**

**SDIO Card Type-A Specification for Bluetooth.**

Applicable Simplified SDA Documents available to non-members and members of the SDA:

**Simplified Version of: SD Memory Card Specification: Part 1 Physical Layer Specification**

**Simplified Version of: SDIO Card Specification:**

**Simplified Version of: SDIO Card Type-A Specification for Bluetooth**

More information on the Secure Digital Association and the SD specifications can be found at the SDA website at. http://www.sdcard.org.

# 4 COMMUNICATION

## 4.1 OVERVIEW

Figure 4.1 below is a diagram of the communication interface between a Bluetooth SD device and the Bluetooth host protocol stack. Modifications to this diagram might be needed for operating systems that do not support a miniport model:

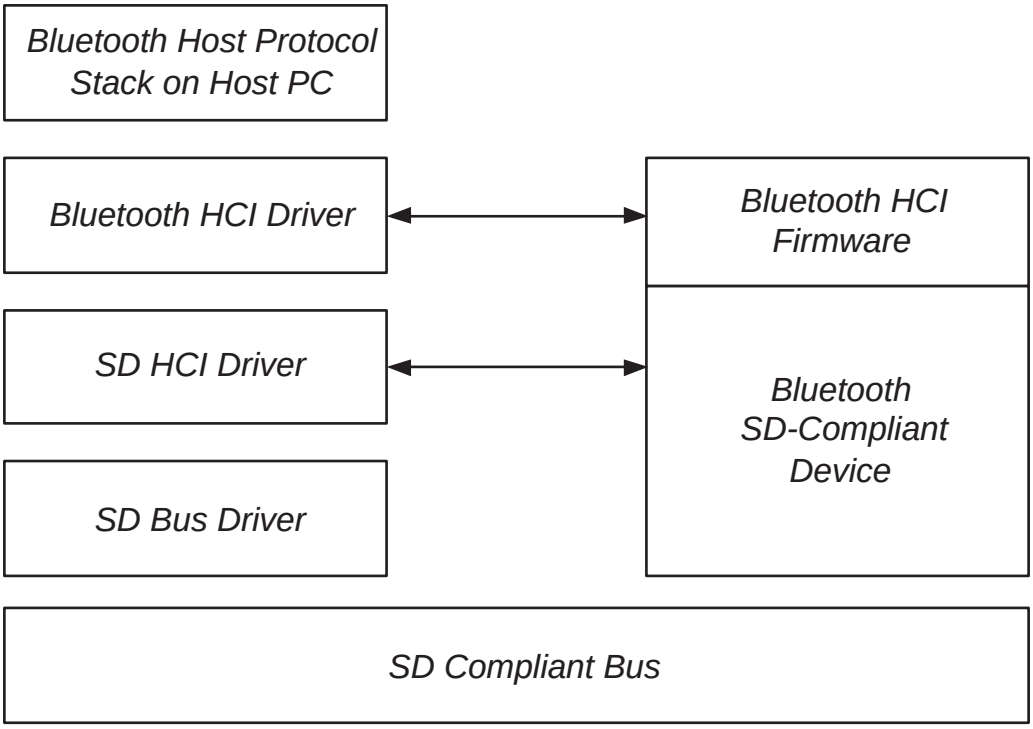



*Figure 4.1: SD Communication Diagram*

# 5 APPENDIX A - ACRONYMS AND ABBREVIATIONS

| Acronym | Description |
|---|---|
| HCI | Host Controller Interface |
| NDA | Non-Disclosure Agreement |
| OS | Operating System |
| SD | Secure Digital |
| SDA | Secure Digital Association |
| SDIO | Secure Digital Input/Output |
| SDP | Service Discovery Protocol |
| SIG | Special Interest Group |

*Table 5.1: Acronyms and Abbreviations*

# 6 APPENDIX B - RELATED DOCUMENTS

A) Bluetooth Core Specification v1.2 or later.

B) Applicable SDA Documents available to members of the SDA:

[1] B.1) SD Memory Card Specification: Part 1 Physical Layer Specification

[2] B.2) SDIO Card Specification

[3] B.3) SDIO Card Type-A Specification for Bluetooth

[4] B.4) SDIO Card Type-B Specification for Bluetooth

[5] B.5) SDIO Card Physical Test Specification

[6] B.5) SDIO Host Physical Test Specification

[7] B.6) SD Bluetooth Type A Test Specification

**These documents are available to members of the SDA in the "Members Only" section of the SDA web site** (https://www.sdcard.org/members/). See http://www.sdcard.org/developers/join/ for information on joining the SDA. C) Applicable Simplified SDA Documents available to non-members and members of the SDA:

C.1) Simplified Version of: SD Memory Card Specification: Part 1 Physical Layer Specification
http://www.sdcard.org/developers/tech/sdcard/pls/

C.2) Simplified Version of: SDIO Card Specification http://www.sdcard.org/developers/tech/sdio/sdio_spec/

C.3) Simplified Version of: SDIO Card Type-A Specification for Bluetooth http://www.sdcard.org/developers/tech/sdio/sd_bluetooth_spec/

# 7 APPENDIX C - TESTS

The SDA has defined formal test procedures for SDIO Type A Bluetooth cards (Controller) and Hosts. It is expected that both Controllers and Hosts will comply with all test requirements set forth by the SDA in accordance with the rules of the SDA. The Bluetooth SIG does not require any formal testing to comply with SIG requirements. The test document names are listed in Appendix B.

## 7.1 TEST SUITE STRUCTURE

There are two types of tests defined for the HCI SD Transport Layer:

1. Functional Tests
2. Protocol Tests

Tests of both types are defined for both the Host and Controller.

The purpose of the functional tests is to verify that the SD Bluetooth Type A Specification, SDIO Standard and SD Physical Standard have been implemented according to the specifications. These tests and the test environment for these tests are defined in documents provided by the SDA.

The purpose of the protocol tests are to verify that the Bluetooth Controller SD implementation or the Host implementation are according to the SD Bluetooth Type A specification.

The test environment for the protocol tests consists of the tester and the Device Under Test (DUT) as illustrated in Figure 7.1 below.

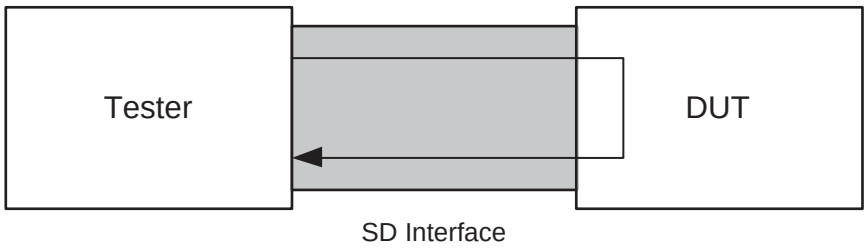



*Figure 7.1: Protocol Test Environment*

The tester is typically a PC with an SD interface. The DUT is placed into local loopback mode and standard HCI commands are used to drive the tests. The test results are verified in the tester.

# THREE-WIRE UART TRANSPORT LAYER

***This document describes the Three-Wire UART transport layer (between the Host and Controller). HCI command, event and data packets flow through this layer, but the layer does not decode them.***


Ford Ex. 1052

# CONTENTS

**1 General ..... 61**

**2 Overview ..... 62**

**3 Slip Layer ..... 63**

3.1 Encoding a Packet ..... 63

3.2 Decoding a Packet ..... 63

**4 Packet Header ..... 65**

4.1 Sequence Number ..... 65

4.2 Acknowledge Number ..... 66

4.3 Data Integrity Check Present ..... 66

4.4 Reliable Packet ..... 66

4.5 Packet Type ..... 66

4.6 Payload Length ..... 67

4.7 Packet Header Checksum ..... 67

**5 Data Integrity Check ..... 68**

5.1 16 Bit CCITT-CRC ..... 68

**6 Reliable Packets ..... 69**

6.1 Header Checksum Error ..... 69

6.2 Slip Payload Length Error ..... 69

6.3 Data Integrity Check Error ..... 69

6.4 Out Of Sequence Packet Error ..... 69

6.5 Acknowledgement ..... 70

6.6 Resending Packets ..... 70

6.7 Example Reliable Packet Flow ..... 70

**7 Unreliable Packets ..... 73**

7.1 Unreliable Packet Header ..... 73

7.2 Unreliable Packet Error ..... 73

**8 Link Establishment ..... 74**

8.1 Uninitialized State ..... 75

8.2 Initialized State ..... 75

8.3 Active State ..... 75

8.4 Sync Message ..... 76

8.5 Sync Response Message ..... 76

8.6 Config Message ..... 76

8.7 Config Response Message ..... 77

8.8 Configuration Field ..... 77

8.8.1 Configuration Messages ..... 78

8.8.2 Sliding Window Size ..... 78

8.8.3 Level of Data Integrity Check ..... 78

8.8.4 Out of Frame Software Flow Control ..... 79
8.8.5 Version Number ..... 79

**9 LOW POWER ..... 80**
9.1 Wakeup Message ..... 80
9.2 Woken Message ..... 80
9.3 Sleep Message ..... 81

**10 Out of Frame Control ..... 82**
10.1 Software Flow Control ..... 82

**11 Hardware Configuration ..... 83**
11.1 Wires ..... 83
11.1.1 Transmit & Receive ..... 83
11.1.2 Ground ..... 83
11.2 Hardware Flow ..... 83
11.2.1 RTS & CTS ..... 83

**12 Recommended Parameters ..... 84**
12.1 Timing Parameters ..... 84
12.1.1 Acknowledgement of Packets ..... 84
12.1.2 Resending Reliable Packets ..... 84

**13 References ..... 85**

# 1 GENERAL

The HCI Three-Wire UART Transport Layer makes it possible to use the Bluetooth HCI over a serial interface between two UARTs. The HCI Three-Wire UART Transport Layer assumes that the UART communication may have bit errors, overrun errors or burst errors. See also “UART Transport Layer” on page 9[vol. 4].

# 2 OVERVIEW

The HCI Three-Wire UART Transport Layer is a connection based protocol that transports HCI commands, events, ACL and Synchronous packets between the Host and the Controller. Packet construction is in done in two steps. First, it adds a packet header onto the front of every HCI Packet which describes the payload. Second, it frames the packets using a SLIP protocol. Finally, it sends this packet over the UART interface.

The SLIP layer converts an unreliable octet stream into an unreliable packet stream. The SLIP layer places start and end octets around the packet. It then changes all occurrences of the frame start or end octet in the packet to an escaped version.

The packet header describes the contents of the packet, and if this packet needs to be reliably transferred, a way of identifying the packet uniquely, allowing for retransmission of erroneous packets.

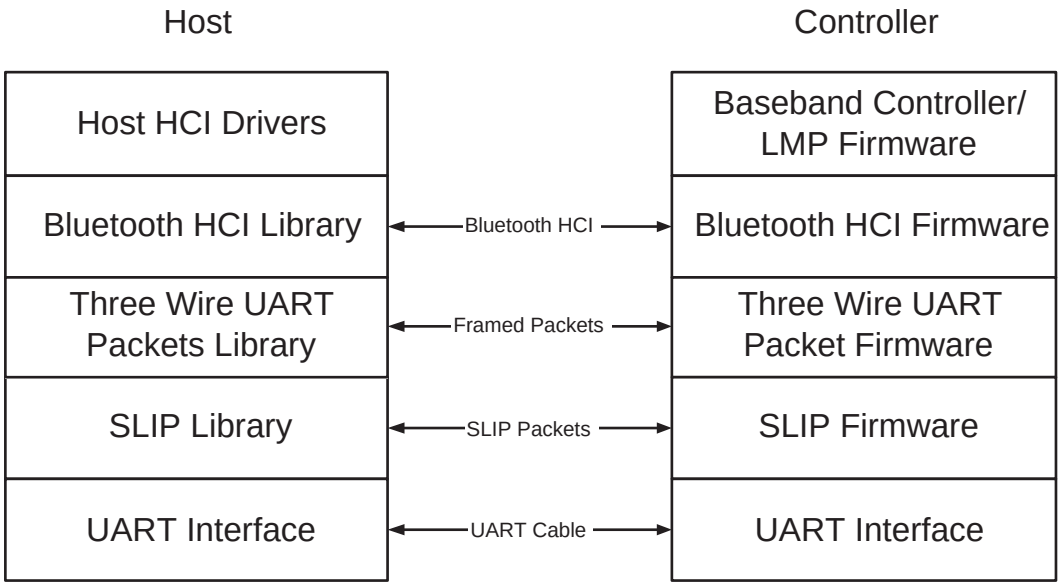



*Figure 2.1: The Relationship Between the Host and the Controller*

# 3 SLIP LAYER

The SLIP layer places packet framing octets around each packet being transmitted over the Three-Wire UART Transport Layer. This delimits the packets and allows packet boundaries to be detected if the receiver loses synchronization. The SLIP layer is based upon the RFC 1055 Standard [1].

## 3.1 ENCODING A PACKET

The SLIP layer performs octet stuffing on the octets entering the layer so that specific octet codes which may occur in the original data do not occur in the resultant stream.

The SLIP layer places octet 0xC0 at the start and end of every packet it transmits. Any occurrence of 0xC0 in the original packet is changed to the sequence 0xDB 0xDC before being transmitted. Any occurrence of 0xDB in the original packet is changed to the sequence 0xDB 0xDD before being transmitted. These sequences, 0xDB 0xDC and 0xDB 0xDD are SLIP escape sequences. All SLIP escape sequences start with 0xDB. All SLIP escape sequences are listed in Table 3.1.

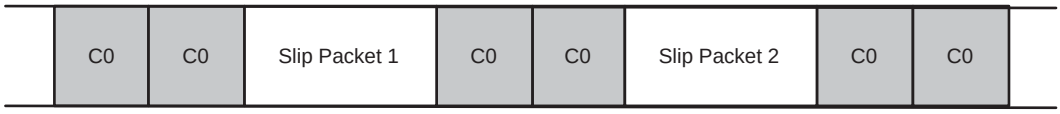



*Figure 3.1: SLIP Packets with 0xC0 at the Start and End of Each Packet*

## 3.2 DECODING A PACKET

When decoding a SLIP stream, a device will first be in an unknown state, not knowing if it is at the start of a packet or in the middle of a packet. The device must therefore discard all octets until it finds a 0xC0. If the 0xC0 is followed immediately by a second 0xC0, then the device will discard the first 0xC0 as it was presumably the end of the last packet, and the second 0xC0 was the start of the next packet. The device will then be in the decoding packet state. It can then decode the octets directly changing any SLIP escape sequences back into their unencoded form. When the device decodes the 0xC0 at the end of the packet, it will calculate the length of the SLIP packet, and pass the packet data into the packet decoder. The device will then seek the next packet. If the device does not receive an 0xC0 for the start of the next packet, then all octets up to and including the next 0xC0 will be discarded.

| SLIP Escape Sequence | Unencoded form | Notes |
|---|---|---|
| 0xDB 0xDC | 0xC0 | |
| 0xDB 0xDD | 0xDB | |
| 0xDB 0xDE | 0x11 | Only valid when OOF Software Flow Control is enabled |
| 0xDB 0xDF | 0x13 | Only valid when OOF Software Flow Control is enabled |

*Table 3.1: SLIP Escape Sequences*

# 4 PACKET HEADER

Every packet that is sent over the Three-Wire UART Transport Layer has a packet header. It also has an optional Data Integrity Check at the end of the payload. The Transport Layer does not support packet segmentation and reassembly. Each transport packet will contain at most one higher layer packet.

A packet consists of a Packet Header of 4 octets, a Payload of 0 to 4095 octets, and an optional Data Integrity Check of 2 octets. See Figure 4.1.

The Packet header consists of a Sequence Number of 3 bits, an Acknowledge Number of 3 bits, a Data Integrity Check Present bit, a Reliable Packet bit, a Packet Type of 4 bits, a Payload Length of 12 bits and an 8 bit Header Checksum. See Figure 4.2.

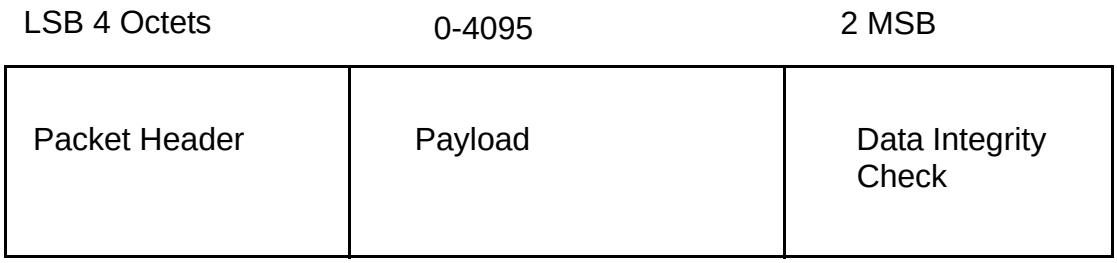



*Figure 4.1: Packet Format*

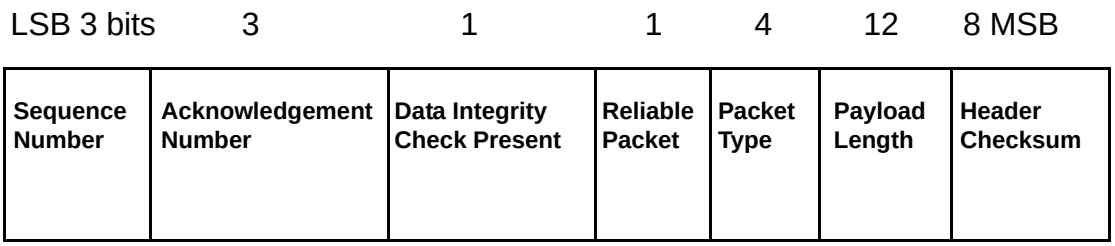



*Figure 4.2: Packet Header Format*

## 4.1 SEQUENCE NUMBER

For unreliable packets this field will be set to 0 on transmit and ignored on receive.

Each new reliable packet will be assigned a sequence number which will be equal to the sequence number of the previous reliable packet plus one modulo eight. A packet will use the same sequence number each time it is retransmitted.

## 4.2 ACKNOWLEDGE NUMBER

The acknowledge number must be set to the sequence number of the next reliable packet this device expects to receive. See Section 6.4 on page 69.

## 4.3 DATA INTEGRITY CHECK PRESENT

If a 16 bit CCITT-CRC Data Integrity Check is appended to the end of the payload, this bit shall be set to 1.

## 4.4 RELIABLE PACKET

If this bit it set to 1, then this packet is reliable. This means that the sequence number field is valid, and the receiving end must acknowledge its receipt. If this bit is set to 0, then this packet is unreliable.

## 4.5 PACKET TYPE

There are four kinds of HCI packets that can be sent via the Three-Wire UART Transport Layer; these are HCI Command Packet, HCI Event Packet, HCI ACL Data Packet and HCI Synchronous Data Packet (see “Host Controller Interface Functional Specification” in the Bluetooth Core Specification v1.2 or later). HCI Command Packets can be sent only to the Controller, HCI Event Packets can be sent only from the Controller, and HCI ACL/ Synchronous Data Packets can be sent both to and from the Controller.

HCI packet coding does not provide the ability to differentiate the four HCI packet types. Therefore, the Packet Type field is used to distinguish the different packets. The acceptable values for this Packet Type field are given in Table 4.1.

| HCI Packet Type | Packet Type |
|---|---|
| Acknowledgement Packets | 0 |
| HCI Command Packet | 1 |
| HCI ACL Data Packet | 2 |
| HCI Synchronous Data Packet | 3 |
| HCI Event Packet | 4 |
| Reserve | 5-13 |
| Vendor Specific | 14 |
| Link Control Packet | 15 |

*Table 4.1: Three-Wire UART Packet Type*

HCI Command Packets, HCI ACL Data Packets and HCI Event Packets are always sent as reliable packets. HCI Synchronous Data Packets are sent as unreliable packets unless HCI Synchronous Flow Control is enabled, in which case they are sent as reliable packets.

In addition to the four HCI packet types, other packet types are defined. One packet type is defined for pure Acknowledgement Packets, and one additional packet type is to support link control. One packet type is made available to vendors for their own use. All other Three-Wire UART Packet Types are reserved for future use.

## 4.6 PAYLOAD LENGTH

The payload length is the number of octets in the payload data. This does not include the length of the packet header, or the length of the optional data integrity check.

## 4.7 PACKET HEADER CHECKSUM

The packet header checksum validates the contents of the packet header against corruption. This is calculated by setting the Packet Header Checksum to a value such that the 2’s complement sum modulo 256 of the four octets of the Packet Header including the Packet Header Checksum is 0xFF.

# 5 DATA INTEGRITY CHECK

The Data Integrity Check field is optional. It can be used to ensure that the packet is valid. The Data Integrity Check field is appended onto the end of the packet. Each octet of the Packet Header and Packet Payload is used to compute the Data Integrity Check.

## 5.1 16 BIT CCITT-CRC

The CRC is defined using the CRC-CCITT generator polynomial

$g(D) = D^{16} + D^{12} + D^5 + 1$

(see Figure 5.1)

The CRC shift register is filled with 1s before calculating the CRC for each packet. Octets are fed through the CRC generator least significant bit first.

The most significant parity octet is transmitted first (where the CRC shift register is viewed as shifting from the least significant bit towards the most significant bit). Therefore, the transmission order of the parity octets within the CRC shift register is as follows:

x[8] (first), x[9],..., x[15], x[0], x[1],..., x[7] (last)

where x[15] corresponds to the highest power CRC coefficient and x[0] corresponds to the lowest power coefficient.

The switch S shall be set in position 1 while the data is shifted in. After the last bit has entered the LFSR, the switch shall be set in position 2, and the registers contents shall be read out for transmission.

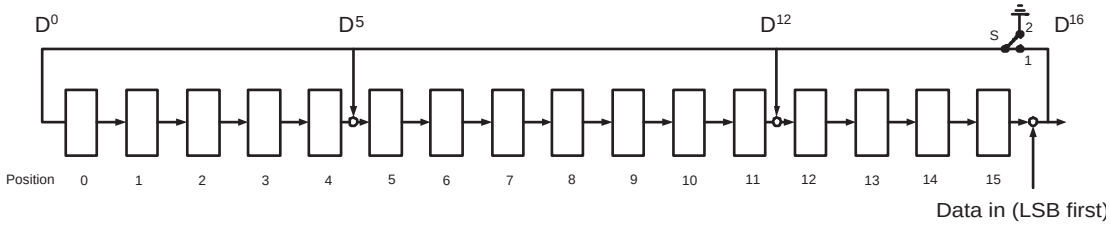



*Figure 5.1: The LFSR Circuit Generating the CRC*

# 6 RELIABLE PACKETS

To allow the reliable transmission of packets through the transport, a method needs to be defined to recover from packet errors. The Host or Controller can detect a number of different errors in the packet.

## 6.1 HEADER CHECKSUM ERROR

The header of the packet is protected by a Packet Header Checksum. If the 2's complement sum modulo 256 of the four octets of the header is not 0xFF, then the packet has an unrecoverable error and all information contained in the packet shall be discarded.

## 6.2 SLIP PAYLOAD LENGTH ERROR

The length of the SLIP packet shall be checked against the Packet Payload Length. If the Data Integrity Check Present bit is set to 1, then the SLIP packet length should be 6 + Packet Payload Length. If the Data Integrity Check Present bit is set to 0, then the SLIP packet length should be 4 + Packet Payload Length. If this check fails, then all information contained in the packet shall be discarded. The SLIP packet length is the length of the data received from the SLIP layer after the SLIP framing, and SLIP escape codes have been processed.

## 6.3 DATA INTEGRITY CHECK ERROR

The packet may have a Data Integrity Check at the end of the payload. This is controlled by the Data Integrity Check Present bit in the header. If this is set to 1, then the Data Integrity Check at the end of the payload is checked. If this is different from the value expected, then the packet shall be discarded. If the link is configured to not use data integrity checks, and a packet is received with the Data Integrity Check Present bit set to 1, then the packet shall be discarded.

## 6.4 OUT OF SEQUENCE PACKET ERROR

Each device keeps track of the sequence number it expects to receive next. This will be one more than the sequence number of the last successfully received reliable packet, modulo eight. If a reliable packet is received which has the expected sequence number, then this packet shall be accepted.

If a reliable packet is received which does not have the expected sequence number, then the packet shall be discarded.

## 6.5 ACKNOWLEDGEMENT

Whenever a reliable packet is received, an acknowledgement shall be generated.

If a packet is available to be sent, the Acknowledgement Number of that packet shall be updated to the latest expected sequence number.

If a requirement to send an acknowledgement value is pending, but there are no other packets available to be sent, the device can send a pure Acknowledgement Packet. This is an Unreliable Packet, with the Packet Type set to 0, Payload Length set to 0, and the Sequence Number set to 0. The Acknowledge Number must be set correctly.

The maximum number of reliable packets that can be sent without acknowledgement defines the sliding window size of the link. This is configured during link establishment. See Sections 8.6, 8.7 and 8.8.

## 6.6 RESENDING PACKETS

A Reliable Packet shall be resent until it is acknowledged. Devices should refrain from resending packets too quickly to avoid saturating the link with retransmits. See Section 12.1.2 on page 84.

## 6.7 EXAMPLE RELIABLE PACKET FLOW

Figure 6.1 shows the transmission of reliable packets between two devices. Device A sends a packet with a Sequence Number of 6, and an Acknowledgement Number of 3. Device B receives this packet correctly, so needs to generate an acknowledgement. Device B then sends a packet with Sequence Number 3 with its Acknowledgement Number set to the next expected packet Sequence Number from Device A of 7.

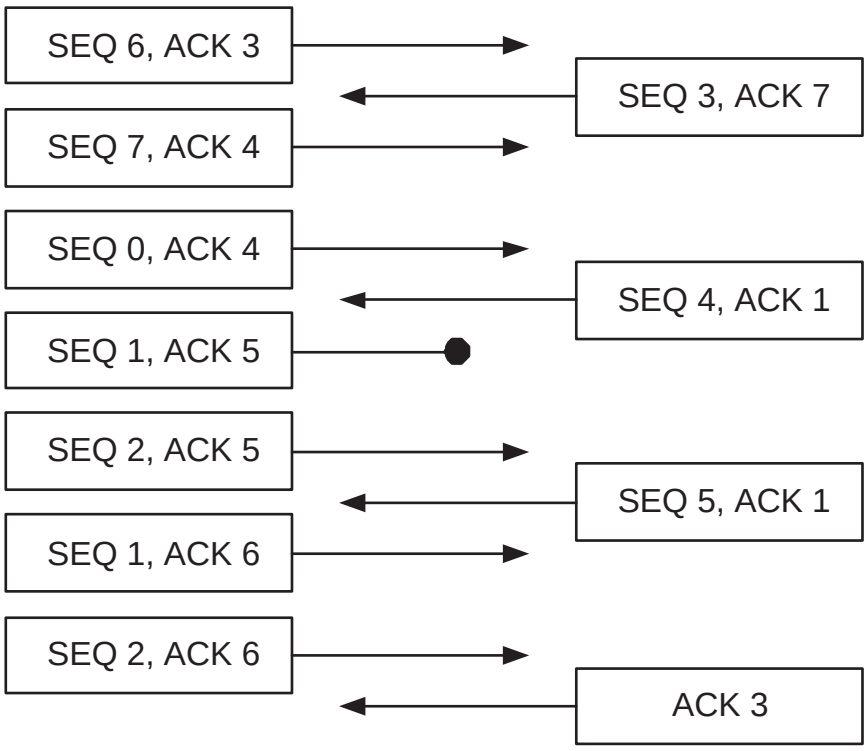



*Figure 6.1: Message Diagram Showing Transmission of Reliable Packets*

Device A receives a packet with Sequence Number 3 and an Acknowledgement Number of 7. Device A was expecting this sequence number so needs to generate an acknowledgement. The Acknowledgement Number of 7 is one greater than the last Sequence Number that was sent, meaning that this packet was received correctly (see Section 6.6 on page 70).

Device A sends two packets, Sequence Numbers 7 and 0. Both packets have the Acknowledgement Number of 4, the next sequence number it expects from Device B. Device B receives the first correctly, and increments its next expected sequence number to 0. It then receives the second packet correctly, and increments the next expected sequence number to 1.

Device B sends a packet with Sequence Number 4, and the Acknowledgement Number of 1. This will acknowledge both of the previous two packets sent by Device A.

Device A now sends two more packets, Sequence Numbers 1 and 2. Unfortunately, the first packet is corrupted. Device B receives the first packet, and discovers the error, so discards this packet (see Section 6.1 on page 69, Section 6.2 on page 69 or Section 6.3 on page 69). It must generate an acknowledgement of this erroneously received reliable packet. Device B then receives the second packet. This is received out of sequence, as it is currently expecting Sequence Number 1, but has received Sequence Number 2 (see 6.4). Again, it must generate an acknowledgement.

Device B sends another packet with Sequence Number 5. It is still expecting a packet with Sequence Number 1 next, so the Acknowledgement Number is set to 1. Device A receives this, and accepts this packet.

Device A has not had either of its last two packets acknowledged, so it must resend them (see 6.6). It does this, but must update the Acknowledgement Number of the original packets that were sent (see Section 6.5 on page 70). The Sequence Numbers of these packets must stay the same (see Section 4.1 on page 65).

Device B receives these packets correctly, and schedules the sending of an acknowledgement. Because Device B doesn't have any data packets that need to be sent, it sends a pure Acknowledgement Packet (see Section 6.5 on page 70).

# 7 UNRELIABLE PACKETS

To allow the transmission of unreliable packets through the transport, the following method shall be used.

## 7.1 UNRELIABLE PACKET HEADER

An unreliable packet header always has the Reliable Packet bit set to 0. The sequence number shall be set to 0. The Data Integrity Check Present, Acknowledgement Number, Packet Type, Payload Length and Packet Header Checksum shall all be set the same as a Reliable Packet.

## 7.2 UNRELIABLE PACKET ERROR

If a packet that is marked as unreliable and the packet has an error, then the packet shall be discarded.

# 8 LINK ESTABLISHMENT

Before any packets except Link Control Packets can be sent, the Link Establishment procedure must be performed. This ensures that the sequence numbers are initialized correctly, it also ensures that the two sides are using the same baud rate, allow detection of peer reset, and allows the device to be configured.

Link Establishment is defined by a state machine with three states: Uninitialized, Initialized and Active. When the transport is first started, the link is in the Uninitialized State. There are four messages that are defined: SYNC, SYNC RESPONSE, CONFIG and CONFIG RESPONSE. All four link establishment messages shall be sent with the Data Integrity Present flag set to 0.

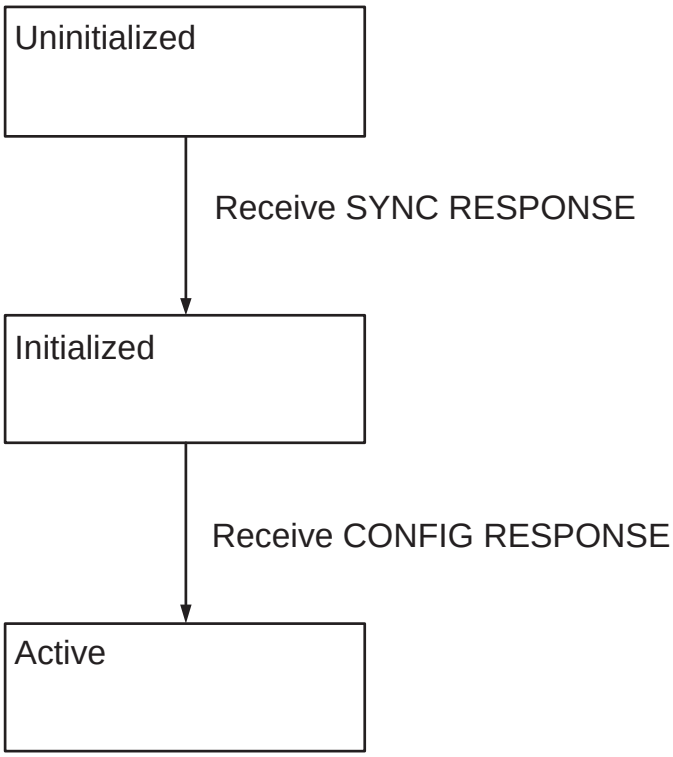



*Figure 8.1: Link Establishment State Diagram*

## 8.1 UNINITIALIZED STATE

In the Uninitialized State a device periodically[1] sends SYNC messages. If a SYNC message is received, the device shall respond with a SYNC RESPONSE message. If a SYNC RESPONSE message is received, the device shall move to the Initialized State. In the Initialized State only SYNC and SYNC RESPONSE messages are valid, all other messages that are received must be discarded. If an invalid packet is received, the device shall respond with a SYNC message. The device shall not send any acknowledgement packets in the Uninitialized State[2].

In the Uninitialized State the Controller may wait until it receives a SYNC message before sending its first SYNC message. This allows the Host to control when the Controller starts to send data.

The SYNC message can be used for automatic baud rate detection. It is assumed that the Controller shall stay on a single baud rate, while the Host could hunt for the baud rate. Upon receipt of a SYNC RESPONSE message, the Host can assume that the correct baud rate has been detected.

## 8.2 INITIALIZED STATE

In the Initialized State a device periodically sends CONFIG messages. If a SYNC message is received, the device shall respond with a SYNC RESPONSE message. If a CONFIG message is received, the device shall respond with a CONFIG RESPONSE message. If a CONFIG RESPONSE message is received, the device will move to the Active State. All other messages that are received must be ignored.

## 8.3 ACTIVE STATE

In the Active State, a device can transfer higher layer packets through the transport. If a CONFIG message is received, the device shall respond with a CONFIG RESPONSE message. If a CONFIG RESONSE message is received, the device shall discard this message.

If a SYNC message is received while in the Active State, it is assumed that the peer device has reset. The local device should therefore perform a full reset of the upper stack, and start Link Establishment again at the Uninitialized State.

Upon entering the Active State, the first packet sent shall have its SEQ and ACK numbers set to zero.

1. During link establishment, various messages are sent periodically. It is suggested to send 4 messages per second.
2. Any packet that was erroneous would normally be acknowledged, as the recipient does not know if the packet was a reliable packet or not. The recipient cannot do this in the Uninitialized State, as it is possible to receive corrupt data while the Uninitialized state.

## 8.4 SYNC MESSAGE

The SYNC message is an unreliable message sent with the Packet Type of 15 and a Payload Length of 2.

The payload is composed of the octet pattern 0x01 0x7E[1].

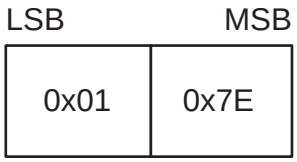



*Figure 8.2: Sync Message Format*

## 8.5 SYNC RESPONSE MESSAGE

The SYNC RESPONSE message is an unreliable message sent with the Packet Type of 15 and a Payload Length of 2. The payload is composed of the octet pattern 0x02 0x7D.

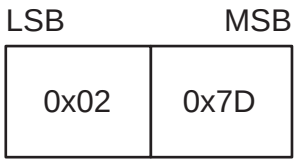



*Figure 8.3: Sync Response Message Format*

## 8.6 CONFIG MESSAGE

The CONFIG message is an unreliable message sent with the Packet Type of 15 and a Payload Length of 2 plus the size of the Configuration Field. The payload is composed of the octet pattern 0x03 0xFC and the Configuration Field.

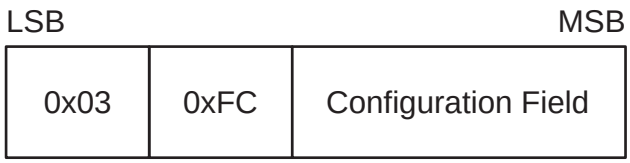



*Figure 8.4: Configuration Message Format*

1. The second octet for all Link Control Packets equals the least significant 7 bits of the first octet, inverted, with the most significant bit set to ensure even parity.

## 8.7 CONFIG RESPONSE MESSAGE

The CONFIG RESPONSE message is an unreliable message sent with the Packet Type of 15 and a Payload Length of 2 plus the size of the Configuration Field. The payload is composed of the octet pattern 0x04 0x7B and the Configuration Field.

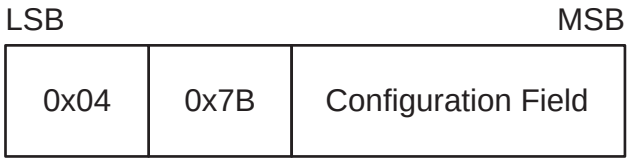



*Figure 8.5: Configuration Response Message Format*

## 8.8 CONFIGURATION FIELD

The Configuration Field contains the Version Number, Sliding Window Size, the Data Integrity Check Type, and if Out Of Frame (OOF) Software Flow Control is allowed.

The Configuration Field in a CONFIG message sent by the Host determines what the Host can transmit and accept. The Configuration Field in a CONFIG RESPONSE message sent by the Controller determines what the Host and Controller shall transmit and can expect to receive.

The Controller sends CONFIG messages without a Configuration Field. The Host sends CONFIG RESPONSE messages without a Configuration Field.

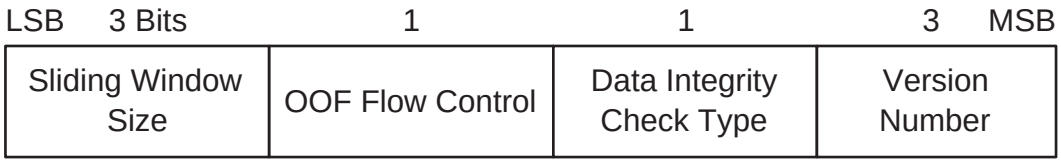



*Figure 8.6: Configuration Field Detail*

To allow for future extension of the Configuration Field, the size of the message determines the number of significant Configuration Octets in the payload. Future versions of this specification may use extra octets. Any bits that are not included in the message shall be set to 0. Any bits that are not defined are reserved and shall be set to 0.

A device shall not change the values if sends in the Configuration Field during Link Establishment.

### 8.8.1 Configuration Messages

The CONFIG – CONFIG RESPONSE message sequence configures the link in both directions. Until a CONFIG RESPONSE message is received only unreliable Link Establishment messages may be sent. Once CONFIG RESPONSE message has been received all other packet types may be sent, and received messages passed up to the Host.

The CONFIG and CONFIG RESPONSE messages contain a set of options for both devices on the link. The Host sends a CONFIG message with the set of options that the Host would like to use. The Controller

responds with a CONFIG RESPONSE message with the set of options that the Host and the Controller will use. This means that the Controller is in full control of the set of options that will be used for all messages sent by both the Host and Controller.

### 8.8.2 Sliding Window Size

This is the maximum number of reliable packets a sender of the CONFIG message can send without requiring an acknowledgement. The value of this field shall be in the range one to seven. The value in the CONFIG RESPONSE message shall be less than or equal to the value in the CONFIG message. For example, the Host may suggest a window size of five in its CONFIG message and the Controller may respond with a value of three in its CONFIG RESPONSE message, but not six or seven. Both devices will then use a maximum sliding window size of three.

### 8.8.3 Level of Data Integrity Check

The CONFIG message contains a bit field describing the types of Data Integrity Checks the sender is prepared to transmit. The peer will select the one it is prepared to use and send its choice in the CONFIG RESPONSE message.

If data integrity checks are not required, then the Data Integrity Check Present bit shall be set to 0 by the Host and Controller.

| Level of Data Integrity | Parameter Description for CONFIG Message |
|---|---|
| 0 | No Data Integrity Check is supported. |
| 1 | 16 bit CCITT-CRC may be used. |

*Figure 8.7: Data Integrity Check Type in the CONFIG Message*

| Level of Data Integrity | Parameter Description for CONFIG RESPONSE Message |
|---|---|
| 0 | No Data Integrity Check must be used. |
| 1 | 16 bit CCITT-CRC may be used. |

*Figure 8.8: Data Integrity Check Type in the CONFIG RESPONSE Message*

### 8.8.4 Out of Frame Software Flow Control

By default, the transport uses no flow control except that mandated by the HCI Functional Specification and the flow control achieved by not acknowledging reliable Host messages. If Software Flow Control is to be used, this needs to be negotiated.

The CONFIG message specifies whether the sender of the CONFIG message is prepared to receive Out of Frame Software Flow Control messages. The CONFIG RESPONSE message specifies whether the peer can send Out of Frame Software Flow Control messages. The CONFIG RESPONSE message may have the field set to 1 only if the CONFIG message had it set to 1. (See Section 10.1 on page 82)

### 8.8.5 Version Number

The Version Number of this protocol shall determine which facilities are available to be used.

The CONFIG message specifies the Version Number supported by the Host. The CONFIG RESPONSE message specifies the Version Number that shall be used by the Host and Controller when sent by the Controller. The value in the CONFIG RESPONSE message shall be less than or equal to the value in the CONFIG message. The Version Numbers are enumerated in Figure 8.9. This specification is version 1.0 (Version Number = 0).

| Version Number | Parameter Description for CONFIG and CONFIG RESPONSE Message |
|---|---|
| 0 | Version 1.0 of this Protocol |
| 1-7 | Reserved for future use |

*Figure 8.9: Version Number in the CONFIG and CONFIG RESPONSE message*

# 9 LOW POWER

After a device is in the Active State, either side of the transport link may wish to enter a low power state. Because recovery from a loss of synchronization is possible, it is allowable to stop listening for incoming packets at any time.

To make the system more responsive after a device has entered a low power state, a system of messages is employed to allow either side to notify the other that they are entering a low power state and to wake a device from that state. These messages are sent as Link Control Packets. It is optional for a device to support the Sleep message. The Wakeup and Woken messages are mandatory.

## 9.1 WAKEUP MESSAGE

The Wakeup message shall be the first message sent whenever the device believes that the other side is asleep. The device shall then repeatedly send the Wakeup message until the Woken message is received. There must be at least a one character gap between the sending of each Wakeup message to allow the UART to resynchronize. The Wakeup message is an unreliable message sent with a Packet Type of 15, and a Payload Length of 2. The payload is composed of the octet pattern 0x05 0xFA. The Wakeup message shall be used after a device has sent a Sleep message. It is mandatory to respond to the Wakeup message.

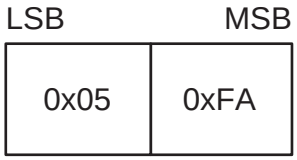



*Figure 9.1: Wakeup Message Payload Format*

## 9.2 WOKEN MESSAGE

The Woken message shall be sent whenever a Wakeup message is received even if the receiver is currently not asleep. Upon receiving a Woken message, a device can determine that the other device is not in a low power state and can send and receive data. The Woken message is an unreliable message sent with a Packet Type of 15, and a Payload Length of 2. The payload is composed of the octet pattern 0x06 0xF9.

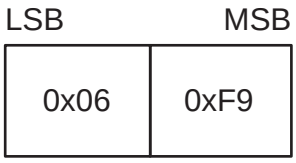



*Figure 9.2: Woken Message Payload Format*

## 9.3 SLEEP MESSAGE

A Sleep message can be sent at any time after Link Establishment has finished. It notifies the other side that this device is going into a low power state, and that it may also go to sleep. If a device sends a Sleep message it shall use the Wakeup / Woken message sequence before sending any data. If a device receives a Sleep message, then it should use the Wakeup / Woken message sequence before sending any data. The Sleep message is an unreliable message sent with a Packet Type of 15, and a Payload Length of 2. The payload is composed of the octet pattern 0x07 0x78.

The sending of this message is optional. The receiver of this message need not go to sleep, but cooperating devices may be able to schedule sleeping more effectively with this message.

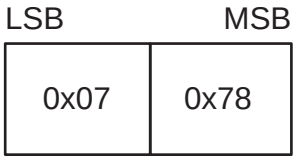



*Figure 9.3: Sleep Message Payload Format*

# 10 OUT OF FRAME CONTROL

It is possible to embed information in the SLIP data stream after a SLIP ESCAPE character that can allow for Software Flow Control. This feature is optional and must be negotiated in the Link Establishment configuration messages.

## 10.1 SOFTWARE FLOW CONTROL

If Software Flow Control is enabled, then the standard XON / XOFF (0x11 and 0x13) characters will control the flow of data over the transport. To allow the XON / XOFF characters to be sent in the payload, they shall be escaped as follows: 0x11 shall be changed to 0xDB 0xDE, 0x13 shall be changed to 0xDB DF. This means that the XON / XOFF characters in the data stream are used only by software flow control.

If Software Flow Control is disabled, then the SLIP escape sequences 0xDB 0xDE and 0xDB 0xDF are undefined. In this case, the original octets of 0x11 and 0x13 shall not be changed. Flow control should always be provided by the tunneled protocols, e.g. HCI Flow Control. Flow control is still available using the standard Sequence Number / Acknowledge Number. This can be done by not acknowledging packets until traffic can resume.

# 11 HARDWARE CONFIGURATION

The HCI Three-Wire UART Transport uses the following configurations.

## 11.1 WIRES

There are three wires used by the HCI Three-Wire UART Transport. These are Transmit, Receive, and Ground.

### 11.1.1 Transmit & Receive

The transmit line from one device shall be connected to the receive line of the other device.

### 11.1.2 Ground

A common ground reference shall be used.

## 11.2 HARDWARE FLOW

Hardware flow control may be used. The signaling shall be the same as a standard RS232 flow control lines. If used, the signals shall be connected in a null-modem fashion; for example, the local RTS shall be connected to the remote CTS and vice versa.

### 11.2.1 RTS & CTS

Request to Send indicates to the remote side that the local device is able to accept more data.

Clear to Send indicates if the remote side is able to receive data.

(See ITU.T recommendations V.24 [2] and V.28 [3])

# 12 RECOMMENDED PARAMETERS

## 12.1 TIMING PARAMETERS

Because this transport protocol can be used with a wide variety of baud rates, it is not possible to specify a single timing value. However, it is possible to specify the time based on the baud rate in use. If $T_{max}$ is defined as the maximum time in seconds it will take to transmit the largest packet over this transport, $T_{max}$ can be expressed as:

$T_{max}$ = maximum size of a packet in bits / baud rate

The maximum size of a packet in bits is either the number of bits in a 4095 octet packet (32,760) or less if required in an embedded system or as determined by the Host or Controller[1]. Thus, at a baud rate of 921,600 and the maximum packet size of 4095 octets, $T_{max}$ is: (4095*10) / 921,600 = 44.434ms.

### 12.1.1 Acknowledgement of Packets

It is not necessary to acknowledge every packet with a pure acknowledgement packet if there is a data packet that will be sent soon. The recommended maximum time before starting to send an acknowledgement is 2 * $T_{max}$.

### 12.1.2 Resending Reliable Packets

A reliable packet must be resent until it is acknowledged. The recommended time between starting to send the same packet is 3 * $T_{max}$.

1. This can be determined using the HCI_Read_Buffer_Size command.

# 13 REFERENCES

[1] [IETF RFC 1055: Nonstandard for transmission of IP datagrams over serial lines: SLIP – http://www.ietf.org/rfc/rfc1055.txt

[2] ITU Recommendation V.24: List of definitions for interchange circuits between data terminal equipment (DTE) and data circuit-terminating equipment (DCE) – http://www.itu.int/rec/recommendation.asp

[3] ITU Recommendation V.28: Electrical characteristics for unbalanced double-current interchange circuits – http://www.itu.int/rec/recommendation.asp



Specification Volume 5

# SPECIFICATION OF THE *BLUETOOTH* SYSTEM

Experience More



# Core System Package

## [AMP Controller volume]

Covered Core Package version:
4.0

Bluetooth®



Ford Ex. 1052

## Revision History

The Revision History is shown in the [Vol. 0, Part C] , Appendix on page 55.

## Contributors

The persons who contributed to this specification are listed in [Vol. 0, Part C] , Appendix on page 55 Appendix on page 55.

## Web Site

This specification can also be found on the official Bluetooth web site:
http://www.bluetooth.com

## Disclaimer and Copyright Notice

The copyright in these specifications is owned by the Promoter Members of Bluetooth SIG, Inc. (“Bluetooth SIG”). Use of these specifications and any related intellectual property (collectively, the “Specification”), is governed by the Promoters Membership Agreement among the Promoter Members and Bluetooth SIG (the “Promoters Agreement”), certain membership agreements between Bluetooth SIG and its Adopter and Associate Members (the “Membership Agreements”) and the Bluetooth Specification Early Adopters Agreements (“1.2 Early Adopters Agreements”) among Early Adopter members of the unincorporated Bluetooth special interest group and the Promoter Members (the “Early Adopters Agreement”). Certain rights and obligations of the Promoter Members under the Early Adopters Agreements have been assigned to Bluetooth SIG by the Promoter Members.

Use of the Specification by anyone who is not a member of Bluetooth SIG or a party to an Early Adopters Agreement (each such person or party, a “Member”), is prohibited. The legal rights and obligations of each Member are governed by their applicable Membership Agreement, Early Adopters Agreement or Promoters Agreement. No license, express or implied, by estoppel or otherwise, to any intellectual property rights are granted herein.

Any use of the Specification not in compliance with the terms of the applicable Membership Agreement, Early Adopters Agreement or Promoters Agreement is prohibited and any such prohibited use may result in termination of the applicable Membership Agreement or Early Adopters Agreement and other liability permitted by the applicable agreement or by applicable law to Bluetooth SIG or any of its members for patent, copyright and/or trademark infringement.

THE SPECIFICATION IS PROVIDED “AS IS” WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, SATISFACTORY QUALITY, OR REASONABLE SKILL OR CARE, OR ANY WARRANTY ARISING OUT OF ANY COURSE OF DEALING, USAGE, TRADE PRACTICE, PROPOSAL, SPECIFICATION OR SAMPLE.

Each Member hereby acknowledges that products equipped with the Bluetooth® technology (“Bluetooth® Products”) may be subject to various regulatory controls under the laws and regulations of various governments worldwide. Such laws and regulatory controls may govern, among other things, the combination, operation, use, implementation and distribution of Bluetooth® Products. Examples of such laws and regulatory controls include, but are not limited to, airline regulatory controls, telecommunications regulations, technology transfer controls and health and safety regulations. Each Member is solely responsible for the compliance by their Bluetooth® Products with any such laws and regulations and for obtaining any and all required authorizations, permits, or licenses for their Bluetooth® Products related to such regulations within the applicable jurisdictions. Each Member acknowledges that nothing in the Specification provides any information or assistance in connection with securing such compliance, authorizations or licenses. NOTHING IN THE SPECIFICATION CREATES ANY WARRANTIES, EITHER EXPRESS OR IMPLIED, REGARDING SUCH LAWS OR REGULATIONS.

ALL LIABILITY, INCLUDING LIABILITY FOR INFRINGEMENT OF ANY INTELLECTUAL PROPERTY RIGHTS OR FOR NONCOMPLIANCE WITH LAWS, RELATING TO USE OF THE SPECIFICATION IS EXPRESSLY DISCLAIMED. BY USE OF THE SPECIFICATION, EACH MEMBER EXPRESSLY WAIVES ANY CLAIM AGAINST BLUETOOTH SIG AND ITS PROMOTER MEMBERS RELATED TO USE OF THE SPECIFICATION.

Bluetooth SIG reserves the right to adopt any changes or alterations to the Specification as it deems necessary or appropriate.

Copyright © 1999 - 2010

Ericsson AB,
Lenovo,
Intel Corporation,
Microsoft Corporation,
Motorola, Inc.,
Nokia Corporation,
Toshiba Corporation

*Third-party brands and names are the property of their respective owners.

# CONTENTS


**1 Introduction ..... 14**

1.1 Organization of the 802.11 PAL ..... 14

**2 AMP Host Controller Interface ..... 16**

2.1 Read Local Version Information Command ..... 16
2.2 Read Local Amp Info Command ..... 16
2.3 Reset Command ..... 18
2.4 Read Failed Contact Counter Command ..... 19
2.5 Read Link Quality Command ..... 19
2.6 Read RSSI Command ..... 19
2.7 Short Range Mode Command ..... 19
2.8 Write Best Effort Flush Timeout Command ..... 20
2.9 Read Best Effort Flush Timeout Command ..... 20
2.10 Physical Link Loss Early Warning Event ..... 20
2.11 Physical Link Recovery Event ..... 21
2.12 Channel Selected Event ..... 21
2.13 Short Range Mode Change Completed Event ..... 21
2.14 Data Structures ..... 21
2.14.1 AMP_ASSOC Structure ..... 21
2.14.2 MAC Address ..... 22
2.14.3 802.11 PAL Capabilities ..... 23
2.14.4 Preferred Channel List ..... 24
2.14.5 Connected Channel List ..... 24
2.14.6 802.11 PAL Version ..... 25
2.15 Connection Accept Timeout Configuration Parameter ..... 25

**3 Physical Link Manager ..... 26**

3.1 Physical Link State Machine ..... 26
3.1.1 General rules ..... 26
3.1.2 State Diagram ..... 26
3.1.3 States ..... 27
3.1.4 Events ..... 28
3.1.5 Conditions ..... 28
3.1.6 Actions ..... 29
3.1.7 DISCONNECTED State ..... 29
3.1.8 STARTING State ..... 31
3.1.9 CONNECTING State ..... 31
3.1.10 AUTHENTICATING State ..... 32
3.1.11 CONNECTED State ..... 33

3.1.12 DISCONNECTING State ..... 33
3.2 Channel Selection ..... 34
3.2.1 Overview ..... 34
3.2.2 Regulatory ..... 35
3.2.3 Specification of Channel Identifiers ..... 35
3.3 802.11 Link Creation ..... 37
3.3.1 Starting the AMP Network ..... 37
3.3.2 Establishing the 802.11 Link ..... 38
3.3.3 Address Fields of Data Frames ..... 39
3.3.4 Admission Control ..... 39
3.4 Physical Link Maintenance ..... 39
3.5 Physical Link Security ..... 40
3.5.1 Obtaining Key Material ..... 40
3.5.2 Creating a PTK ..... 40
3.5.3 Using Encryption ..... 41
3.5.4 Refreshing a PTK ..... 41
3.5.5 Transporting Security Handshake Messages ..... 41
3.6 Physical Link Support for QOS ..... 42
3.6.1 QoS Advertisement ..... 42
3.6.2 Negotiation ..... 42
**4 Logical Link Manager ..... 43**
4.1 Logical Link Creation ..... 43
4.1.1 Logical Link Handles ..... 43
4.1.2 Null Traffic Logical Links ..... 43
4.1.3 Best Effort Logical Links ..... 44
4.1.4 Guaranteed Logical Links ..... 44
4.2 Logical Link Updates ..... 44
4.3 Logical Link Deletion ..... 45
**5 Data Manager ..... 46**
5.1 Encapsulation ..... 46
5.2 Coexistence and Local Interference ..... 47
5.2.1 Interference from Collocated Radios ..... 47
5.2.2 Unavailability of Remote Peer ..... 47
5.2.3 Activity Reports ..... 48
5.3 Explicit Flush ..... 50
5.4 Automatic Flush ..... 50
5.5 Quality Of Service Violations ..... 50
**6 Constants ..... 51**
**7 Message Sequence Charts ..... 52**

**8 Appendix A: Test Support .....53**
8.1 AMP Test Command .....53
8.1.1 Test Scenarios .....55
8.1.2 Test Mode Data Frame Format .....56
8.2 AMP Start Test Event .....56
8.3 AMP Test End Event .....57
**9 References .....59**
**Figures .....60**
**Tables .....61**

# 802.11 PROTOCOL ADAPTATION LAYER FUNCTIONAL SPECIFICATION

***This document specifies the Protocol Adaptation Layer for the IEEE 802.11 conformant Alternate MAC/PHY.***


Ford Ex. 1052

# 1 INTRODUCTION

This Part of the Bluetooth Core Specification describes the operation of the Protocol Adaptation Layer (PAL) for a controller incorporating an 802.11 device compliant with the 2007 edition of the IEEE 802.11 Standard (see [1]). In this Part, specific references in [1] will be given by clause number.

The 802.11 PAL defines the protocol state machines, data encapsulation methods, event triggers, and data structures in support of the use of an 802.11 AMP.

## 1.1 ORGANIZATION OF THE 802.11 PAL

To aid understanding of functional descriptions, Figure 1.1 shows the organization of the 802.11 PAL. This structure is informative.

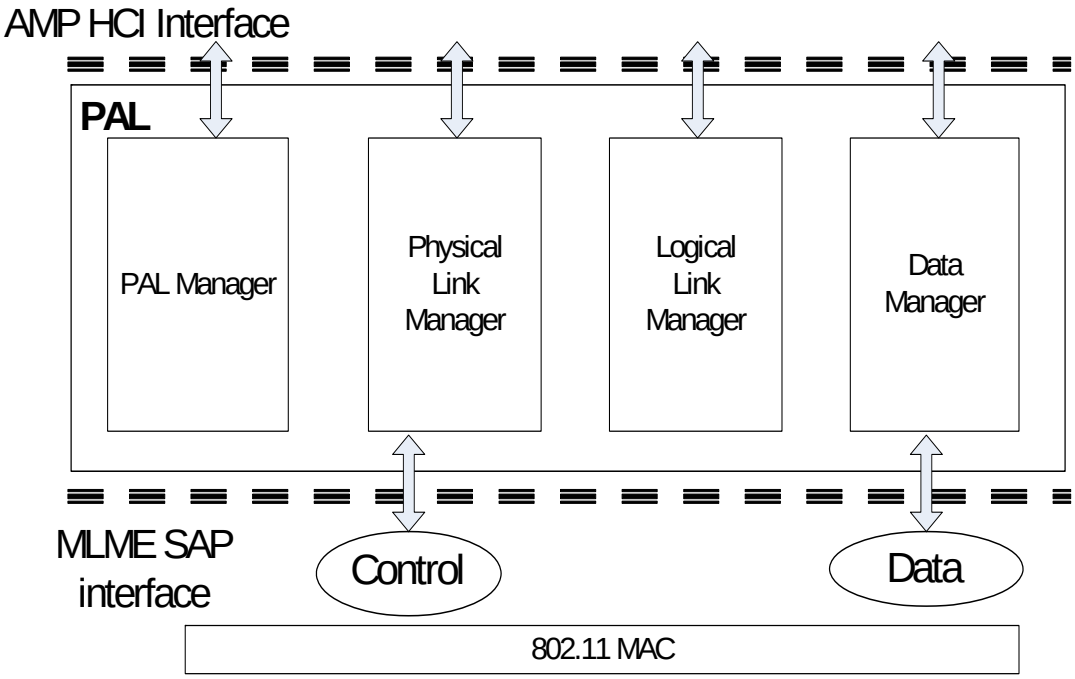



*Figure 1.1: Internal structure of the 802.11 PAL*

The upper edge of the 802.11 PAL provides a single instance of the logical HCI with AMP functionality. The behavior of the PAL is defined at this interface in terms of logical HCI operations. Implementations may optionally use a physical HCI transport.

For clarity of description, the lower edge of the PAL uses services including those defined in [1] clause 10.3.

The PAL Manager implements operations that are global to the PAL. This includes responding to host requests for AMP info and PAL version as well as performing PAL reset.

The Physical Link Manager implements operations on physical links. Physical link semantics are defined in Section 3. Supported operations include physical link creation and acceptance, and deletion of physical links. Supported operations at the MAC interface include PHY channel selection and security establishment and maintenance.

The Logical Link Manager implements operations on logical links. Logical link semantics are defined in Section 4. Each logical link exists with respect to a single physical link. Supported operations include creation and deletion of logical links and changes to QoS parameters for those logical links. At the MAC interface this includes the mapping of extended flow specifications to user priorities.

The Data Manager performs operations on data packets and is described in Section 5. Each data packet is associated with exactly one extended flow specification and therefore exactly one logical link. Supported operations include transmit, receive and buffer management operations such as flush events. At the MAC interface this includes interactions with the MAC transmit and receive operations and determination of the next packet to send on any particular link.

# 2 AMP HOST CONTROLLER INTERFACE

A number of elements used in HCI commands and events are defined to be AMP-type specific. This section describes the values to be used for the 802.11 PAL.

Octet ordering conventions for parameters and fields in this section shall be as defined in HCI, [Vol 2] Part E, Section 5.2.

## 2.1 READ LOCAL VERSION INFORMATION COMMAND

Two return parameter values from this HCI command are defined to be AMP-type specific. For the 802.11 PAL they shall be as follows.

*PAL _Version* *Size: 1 Octet*

| Value | Parameter Description |
|---|---|
| 0xXX | Version of the current PAL in the Controller. See Bluetooth Assigned Numbers. |

*PAL_Sub-version:* *Size: 2 Octets*

| Value | Parameter Description |
|---|---|
| 0xXXXX | In an 802.11 PAL this value is vendor specific. |

## 2.2 READ LOCAL AMP INFO COMMAND

See [Vol 2] Part E, Section 7.5.7.

There are six return parameters for the Read Local AMP Info command which are 802.11 AMP specific.

*Total_Bandwidth:* *Size: 4 Octets*

| Value | Parameter Description |
|---|---|
| 0xXXXXXXXX | An upper bound on the total data rate that can be achieved by the AMP for applications. It accounts for the total bandwidth provided by the HCI transport. No sustained combination of transmit and receive operations shall exceed this value. This may be used to help in AMP selection and admission control. Expressed in kbps.<br>For testing purposes, the achievable throughput deliverable to applications by an ERP PHY shall be assumed to be no more than 30000 kbps. |

*Max_Guaranteed_Bandwidth:* *Size: 4 Octets*

| Value | Parameter Description |
|---|---|
| 0xXXXXXXXX | An upper bound on the maximum data rate as seen by the application that the AMP can guarantee for a logical channel. Any request made by an application above this level would be rejected. It accounts for any bandwidth limitations of the HCI transport. No sustained combination of transmit and receive operations shall exceed this value. This can be used to help in AMP selection and admission control. Expressed in kbps.<br>The Max_Guaranteed_Bandwidth parameter value returned shall be no greater than the Total_Bandwidth parameter. This value is not a guarantee of bandwidth and should be interpreted as an upper bound on the sum of the bandwidths of all active flow specs. |

*Min_Latency:* *Size: 4 Octets*

| Value | Parameter Description |
|---|---|
| | The Min_Latency parameter value is the practical lower bound on the service latency that can be provided by the 802.11 AMP. The lower bound of the service latency is the time from when a frame is issued to the AMP HCI until the MAC starts transmitting the frame with no contention window back off. This shall be equal to the AMP HCI minimum latency + DIFS + CWmin where the DIFS and CWmin are as given in [1] clause 9.2.10. |

*Max_PDU_Size:* *Size: 4 Octets*

| Value | Parameter Description |
|---|---|
| | An upper bound on the size of L2CAP PDU which may be provided for transmission or reception on this AMP. The Host shall not require the AMP to transport L2CAP PDUs larger than this value. Expressed in octets. The Maximum PDU Size parameter described in [[Vol 3] Part A, Section 5.4] for any connection over this AMP should not exceed this value.<br>The Max_PDU_Size parameter returned shall be Max80211PALPDUSize. |

*Controller_Type:* *Size: 1 Octet*

| Value | Parameter Description |
|---|---|
| 0x01 | 802.11 AMP |

*PAL_Capabilities:* *Size: 2 Octets*

| Value | Parameter Description |
|---|---|
| 0xXXXX | Bit 0: "Service Type = Guaranteed" is not supported by PAL = 0<br>"Service Type = Guaranteed" is supported by PAL = 1<br>Bits 15-1: Reserved<br>(See L2CAP, [Vol 3] Part A, Section 5.6) |

Bit 0 of the PAL_Capabilities parameter shall be set to 1 if the local 802.11 AMP device is capable of using EDCA (see [1] clause 9.9.1), otherwise it shall be set to 0.

*AMP_ASSOC_Length:* *Size: 2 Octets*

| Value | Parameter Description |
|---|---|
| 0xXXXX | AMP_ASSOC maximum length for this AMP Controller. |

The 802.11 PAL shall set this to Max80211AMPASSOCLen.

*Max_Flush_Timeout:* *Size: 4 Octets*

| Value | Parameter Description |
|---|---|
| 0xXXXXXXXX | Maximum time period, in microseconds, which the AMP device may use to attempt to transmit a frame on a guaranteed logical link. This value is the sum of the durations of all 802.11 transmission attempts for a given frame. It should be chosen with the expectation that the 802.11 MAC may be denied access to the medium for a given transmission attempt. This may be due to interference from collocated radios, or otherwise.<br>If the Controller is configured to retry frames for an unbounded time (there is no flushing at all), then the PAL shall set this value to 0xFFFFFFFF. |

*Best_Effort_Flush_Timeout:* *Size: 4 Octets*

| Value | Parameter Description |
|---|---|
| 0xXXXXXXXX | The typical time period, in microseconds, which the AMP device may use to attempt to transmit a frame on a best effort logical link. This value is the sum of the duration of all 802.11 transmission attempts for a given frame. It should be chosen with the expectation that the 802.11 MAC is usually able to access the medium for each attempt. The value shall not exceed the value given in Max_Flush_Timeout.<br>If the Controller is configured to retry frames for an unbounded time (i.e. there is no flushing at all), then the PAL shall set this value to 0xFFFFFFFF. |

## 2.3 RESET COMMAND

See [Vol 2] Part E, Section 7.3.2.

In addition to setting the HCI parameters to their default values, when the 802.11 PAL receives an AMP HCI_Reset command it shall destroy all existing AMP physical links. Informative note: Non AMP links should not be destroyed.

## 2.4 READ FAILED CONTACT COUNTER COMMAND

When the 802.11 PAL receives an HCI Read_Failed_Contact_Counter command it shall return the number of consecutive incidents in which the remote device didn't respond after the flush timeout had expired, and the L2CAP packet that was currently being transmitted was automatically flushed. The Failed Contact Counter is specific to each logical link.

## 2.5 READ LINK QUALITY COMMAND

See [Vol 2] Part E, Section 7.5.3.

The meaning of the Link_Quality parameter in the Read Link Quality command is as shown below.

*Link_Quality:* *Size: 1 Octet*

| Value | Parameter Description |
|---|---|
| 0xXX | In an 802.11 AMP this unsigned 8-bit value shall be the Link Quality Indicator value. It shall be 0 if the Link Quality Indicator value is not available.<br>Range: 0x00 ≤ N ≤ 0xFF |

## 2.6 READ RSSI COMMAND

See [Vol 2] Part E, Section 7.5.4.

The meaning of the RSSI parameter in the Read RSSI command is as shown below.

*RSSI:* *Size: 1 Octet*

| Value | Parameter Description |
|---|---|
| 0xXX | This value is a signed 8-bit value, and is interpreted as an indication of arriving signal strength at the antenna measured in dBm.<br>The value shall be 0x81 (-127 dBm) if the signal strength indication is not available. |

## 2.7 SHORT RANGE MODE COMMAND

When the Host determines that the AMP peers may have insufficient separation to obtain full AMP throughput, it may enable Short Range Mode in the PAL. The Short Range Mode command may be used by the Host to indicate to the PAL whether or not to operate in Short Range Mode.

When in Short Range Mode, the PAL shall limit to ShortRangeModePowerMax the transmit power in dBm (measured at the antenna) for all 802.11 AMP ERP-OFDM frames transmitted by the device on the given physical link, as necessary to prevent exceeding the maximum input signal level of the peer. If the Host does not enable Short Range Mode or if the Host sets Short Range Mode to disabled, the PAL shall assume it may set the maximum transmit power for the link as it deems appropriate, respecting regulatory limits. If the AMP device is not able to limit its transmit power due to other extant connections then it may use a transmit power greater than ShortRangeModePowerMax in order to preserve those connections.

The meaning of the Short Range Mode parameter of the HCI_Short_Range_Mode command is as follows:

*Short Range Mode:* *Size: 1 octet*

| Value | Parameter Description |
|---|---|
| 0xXX | 0x00–Short range mode shall be disabled in the PAL (default)<br>0x01–Short range mode shall be enabled in the PAL.<br>0x02...0xFF - Reserved |

When the AMP controller receives the Short_Range_Mode command, it shall indicate a Command Status event. Later, after the MAC programming is completed, the controller shall generate a Short_Range_Mode_Change_Completed event. See Section 2.13.

## 2.8 WRITE BEST EFFORT FLUSH TIMEOUT COMMAND

*Best_Effort_Flush_Timeout:* *Size: 4 octets*

| Value | Parameter Description |
|---|---|
| 0xXXXXXXXX | 0x00000000–0xFFFFFFFE: Best Effort Flush Timeout value in microseconds.<br>0xFFFFFFFF–No Best Effort Flush Timeout used. (default) |

## 2.9 READ BEST EFFORT FLUSH TIMEOUT COMMAND

*Best_Effort_Flush_Timeout:* *Size: 4 octets*

| Value | Parameter Description |
|---|---|
| 0xXXXXXXXX | 0x00000000–0xFFFFFFFE: Best Effort Flush Timeout value in microseconds.<br>0xFFFFFFFF–No Best Effort Flush Timeout used (default) |

## 2.10 PHYSICAL LINK LOSS EARLY WARNING EVENT

Implementation of this event is not required for 802.11 AMPs.

## 2.11 PHYSICAL LINK RECOVERY EVENT

Implementation of this event is not required for 802.11 AMPs.

## 2.12 CHANNEL SELECTED EVENT

See [Vol 2] Part E, Section 7.7.52.

When an HCI Channel Selected event is indicated by the PAL with successful status, it signifies the local 802.11 MAC has been configured to start operating on the selected channel.

Subsequent to the HCI Channel Selected event, the initiating AMP device shall create an AMP_ASSOC containing its MAC address, the 802.11 PAL capabilities, and with only the selected channel in its preferred channel list. The host may obtain this AMP_ASSOC by issuing one or more HCI_Read_Local_AMP_ASSOC commands.

## 2.13 SHORT RANGE MODE CHANGE COMPLETED EVENT

See [Vol 2] Part E, Section 7.7.60.

After the PAL is notified of a change of state in Short Range Mode, it shall program the 802.11 device accordingly, unless the exceptions in Section 2.7 are in effect. When it has finished making such changes to the MAC configuration, or if the PAL has changed the state of the Short Range Mode autonomously, the PAL shall indicate this to the Host by indicating the Short_Range_Mode_Change_Event. The Short_Range_State parameter identifies the new configuration to the Host.

## 2.14 DATA STRUCTURES

### 2.14.1 AMP_ASSOC Structure

The AMP_ASSOC is an AMP type specific structure and appears in various HCI commands and events. The AMP_ASSOC structure used by the 802.11 PAL shall be composed of Type-Length-Value (TLV) triplets.

The general format of such a TLV, shown in Table 2.1, shall be a one-octet TypeID field, a two-octet Length field, and a variable length Value field. The length of the Value field in octets shall be exactly equal to the unsigned number represented by the Length field. A TLV with zero in its Length field shall contain no Value field. If an implementation does not have support for a triplet in a received AMP_ASSOC, it shall ignore the triplet and continue processing any remaining triplets.

The Length field is 2 octets in length and shall be ordered in the AMP_ASSOC according to Volume 2, [Part B] Section 6.2 on page 109. The Value field shall be interpreted as a stream of octets.

The TypeID of 0xFF shall be reserved for use in debugging.

| TypeID | Length | Value |
|---|---|---|
| 1 octet | 2 octets | Variable number of octets |

*Table 2.1: TLV format*

The set of defined TypeIDs is given in Table 2.2.

| TypeID codepoint | Description | AMP_ASSOC inclusion |
|---|---|---|
| 0x00 | Reserved | NA |
| 0x01 | MAC address | Mandatory |
| 0x02 | Preferred channel list | Mandatory |
| 0x03 | Connected channel | Optional |
| 0x04 | 802.11 PAL Capabilities list | Optional |
| 0x05 | 802.11 PAL version | Mandatory |
| 0x06 - 0xFE | Reserved | NA |
| 0xFF | Reserved for use in debugging | NA |

*Table 2.2: TypeIDs used for 802.11 AMP TLVs*

### 2.14.2 MAC Address

The PAL shall use the following format to report the IEEE MAC address of its local 802.11 MAC. The bit ordering of the address is given in [1] clause 7.1.1.

*MAC_Address_TypeID:* *Size: 1 octet*

| Value | Parameter Description |
|---|---|
| 0x01 | MAC address TypeID |

*MAC_Address_Length:* *Size: 2 octets*

| Value | Parameter Description |
|---|---|
| 0x0006 | MAC address Length |

*MAC_Address_Specifier:* *Size: 6 octets*

| Value | Parameter Description |
|---|---|
| 0xXXXXXXXXXXXX | MAC address specifier |

### 2.14.3 802.11 PAL Capabilities

The 802.11 PAL Capabilities is a bit field of supported capabilities of the sending device.

*802.11_PAL_Capabilities_TypeID:* *Size: 1 octet*

| Value | Parameter Description |
|---|---|
| 0x04 | 802.11 PAL Capabilities TypeID |

*802.11_PAL_Capabilities_Length:* *Size: 2 octets*

| Value | Parameter Description |
|---|---|
| 0x0004 | 802.11 PAL Capabilities Length |

*802.11_PAL_Capabilities_Specifier:* *Size: 4 octets*

| Bit format | Parameter Description |
|---|---|
| Bit 0 | When set, signifies PAL capable of utilizing received Activity Reports |
| Bit 1 | When set, signifies PAL is capable of utilizing scheduling information received in an Activity Report |
| Bits 2..31 | Reserved |

The 802.11 PAL Capabilities TLV is optional to include in the AMP_ASSOC. If an 802.11 PAL Capabilities TLV does not appear in an AMP_ASSOC, then the receiver shall interpret this as receiving an 802.11 PAL Capabilities TLV with a Value field containing the default value of all zeros.

*Preferred_Channel_List_TypeID:* *Size: 1 octet*

| Value | Parameter Description |
|---|---|
| 0x02 | Preferred Channel List TypeID |

*Preferred_Channel_List_Length:* *Size: 2 octets*

| Value | Parameter Description |
|---|---|
| N | Length of Preferred Channel List |

*Preferred_Channel_List_Specifier:* *Size: variable*

| Value | Parameter Description |
|---|---|
| 0xXXXXXXXX | Modified 802.11 Country Information element containing list of preferred 802.11 channels. See Section 3.2.3 for details. |

### 2.14.4 Preferred Channel List

The Preferred Channel List is a non-empty list of channels supported and usable by the PAL. The receiver of the Preferred Channel List shall interpret the contents of the list with the assumption that the list is arranged in order of most preferred channel to least preferred channel. The format of the list is identical to the 802.11 Country information element, excluding the 802.11 information, element identifier, length, and pad fields. See [1] clause 7.3.2.9.

### 2.14.5 Connected Channel List

The Connected Channel List specifies the channels which may currently be in use by the sending device. If multiple channels are listed, the ordering gives no implied preference. The format of the list is the same as the 802.11 Country information element, without the element identifier, length, and pad fields. See [1] clause 7.3.2.9.

*Connected_Channel_List_TypeID:* *Size: 1 octet*

| Value | Parameter Description |
|---|---|
| 0x03 | Connected Channel List TypeID |

*Connected_Channel_List_Length:* *Size: 2 octets*

| Value | Parameter Description |
|---|---|
| 0xXXXX | Length of Connected Channel List |

*Connected_Channel_List_Specifier:* *Size: variable*

| Value | Parameter Description |
|---|---|
| 0xXXXXXXXX | Modified 802.11 Country Information element containing list of 802.11 channels. See Section 3.2.3 for details. |

The Connected Channel TLV is optional to include in the AMP_ASSOC. If it is not included, then the receiver shall assume there are no channels which are currently in use by the sending device.

*802.11_PAL_Version_TypeID:* *Size: 1 octet*

| Value | Parameter Description |
|---|---|
| 0x05 | 802.11 PAL Version TypeID |

*802.11_PAL_Version_Length:* *Size: 2 octets*

| Value | Parameter Description |
|---|---|
| 0x0005 | Length of 802.11 PAL Version specifier |

*802.11_PAL_Version_Specifier:* *Size: 1 octet*

| Value | Parameter Description |
|---|---|
| 0xXX | PAL Version |

*802.11_PAL_Company_Identifier:* *Size: 2 octets*

| Value | Parameter Description |
|---|---|
| 0xXXXX | SIG Company identifier of 802.11 PAL vendor |

*802.11_PAL_Sub_Version:* *Size: 2 octets*

| Value | Parameter Description |
|---|---|
| 0xXXXX | PAL Sub-version specifier |

### 2.14.6 802.11 PAL Version

An AMP endpoint may need to discover the version of the remote PAL. The information in the 802.11 PAL Version TLV shall be composed of the PAL_Version from the HCI_Read_Local_Version_Information command, the Bluetooth SIG Company Identifier (see Assigned Numbers, [4]) for the provider of the PAL, and the PAL_Sub-version from the HCI_Read_Local_Version_Information command. The Company Identifier and PAL Sub-version parameters shall use the octet ordering as given in Volume 2, [Part B] Section 6.2 on page 109.

## 2.15 CONNECTION ACCEPT TIMEOUT CONFIGURATION PARAMETER

See [Vol 2] Part E, Section 6.7.

The default value of the Connection Accept Timeout used by the 802.11 PAL shall be 5 seconds.

# 3 PHYSICAL LINK MANAGER

A physical link joins an initiating device and a responding device. The initiating device is the device on which the HCI_Create_Physical_Link command was issued. The responding device is the one on which the HCI_Accept_Physical_Link command was issued. Physical link creation collisions are resolved at a higher level, by the AMP Manager.

Support for the 802.11 ERP (see clause 19 of [1]) shall be mandatory for 802.11 AMP devices, but other PHY types may be supported. Channels 1 (2412 MHz) through 11 (2462 MHz) shall be supported for interoperability.

A physical link represents a transport between a single local 802.11 AMP device and a single remote device with a matching 802.11 AMP. The AMP may support multiple physical links (representing different remote devices) at one time. There is a unique binding between the 802.11 MAC addresses of the devices and the physical link.

For a physical link to exist in the CONNECTED state the two devices must have established a PTKSA as described in [1] clause 8.4.1.

## 3.1 PHYSICAL LINK STATE MACHINE

### 3.1.1 General rules

The behavior of the PAL with respect to physical links is described in terms of a finite state machine. The state machine describes the behavior of the PAL for an individual physical link; extension to multiple physical links is outside the scope of this document. The sequence of external stimulus and behavior at the logical HCI shall be as though this state machine is present in the implementation. Similarly, the sequence of stimulus and behavior at the 802.11 radio interface shall be as though this state machine is present in the implementation.

### 3.1.2 State Diagram

The possible state events and transitions are summarized in this diagram. The diagram itself is Informative.

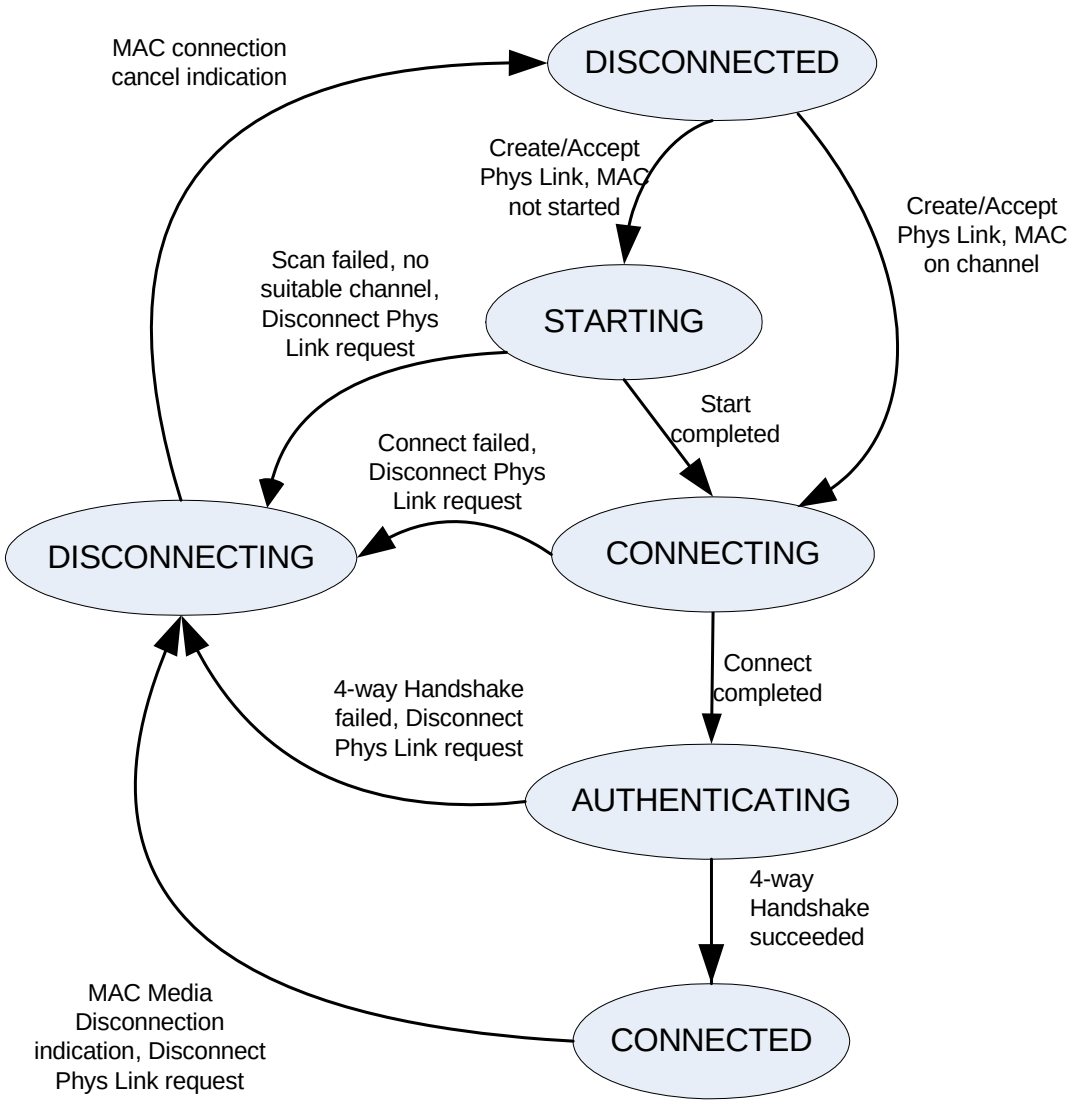



*Figure 3.1: Physical Link finite state machine diagram*

### 3.1.3 States

The following states have been defined to clarify the protocol. The states apply to an individual physical link. At power-up or reset, no physical links exist and the state is the DISCONNECTED state.

DISCONNECTED—The physical link is not active, and this is the initial state.

STARTING—Channel has been selected, the MAC is initializing.

CONNECTING—The initiating device waits for messages from the peer device; the responding device commences network connection operations.

AUTHENTICATING—The devices perform the security association process.

CONNECTED—A secure physical link has been established with the remote device.

DISCONNECTING—The PAL waits for the MAC to complete disconnection and return to the initial state.

### 3.1.4 Events

The following events may cause transitions in the state diagram.

**Create/Accept Physical Link**—HCI commands from the local host which affect the state of a physical link.

**Connection Accept Timeout**—The timeout for this physical link create/accept has expired.

**MAC Start Completed**—The MAC has started operation on the given channel. This includes beaconing and listening for connections.

**MAC Start Failed**—The MAC has failed to start on the specified channel for any reason.

**MAC Connect Completed**—The MAC has completed the process of connecting to a peer in a specified channel.

**MAC Connect Failed**—The MAC fails to connect on the selected channel for any reason

**MAC Media Disconnection Indication**—MAC has signaled an existing connection is lost to a remote device.

**MAC Connection Cancel Indication**—MAC has completed the request to cancel a prior connection.

**HCI Disconnect Physical Link Request**—The host has given a disconnect request to the PAL.

**4-way Handshake Fails**—The establishment of a secure link for this physical link has failed.

**4-way Handshake Succeeds**—The establishment of a secure link for this physical link has completed successfully.

### 3.1.5 Conditions

The following conditions are potential qualifiers for actions and state transitions to occur.

**MAC not yet started in selected channel**—The MAC is not beaconing in any PHY channel, or is beaconing in a different channel than the one specified.

**MAC already in selected channel**—The MAC is already beaconing in the selected PHY channel.

**No suitable channel**—No suitable channel can be selected, or the selected channel cannot be joined for any reason.

**Device is initiator**—Role of device is physical link initiator due to reception of Physical Link Create command.

**Device is responder**—Role of device is physical link responder due to reception of Physical Link Accept command.

### 3.1.6 Actions

In some cases, a state transition causes one or more of the following actions to occur. The actions for a state transition shall occur in their entirety before any subsequent state transition occurs, as follows.

**Determine selected channel**—Using information from the AMP_ASSOC of the remote device and preferences from the local device, select a channel

**Set or clear Connection Accept Timeout** —Start or stop the timer for the current physical link.

**Set or clear NeedPhysLinkCompleteEvent, DiscRequested** —Set or clear state variables which control subsequent behavior.

**Set or PhysLinkCompleteStatus**—Sets command status to be indicated to Host.

**Signal MAC to start on channel**—Command the MAC to start the process of connection on the specified channel.

**Issue MAC connection command**—Command the MAC to attempt to connect to the remote device.

**Initiate 4-way handshake**—Command the authenticator to send the first security message.

**Send HCI event**—send the indicated HCI event to the local Host.

**Cancel MAC connect operation**—This physical link no longer needs to be present on this channel. The PAL signals the MAC to delete the connection.

**Signal MAC to disconnect peer**—Command MAC to send disconnection frame to peer.

### 3.1.7 DISCONNECTED State

This is the initial state. No timers shall be active.

| Event | Condition | Action | Next State |
|---|---|---|---|
| HCI_Create_Physical_Link command | MAC not yet in selected channel | Determine selected channel<br>Request MAC to start on channel<br>Set Connection Accept timeout<br>Set NeedPhysLinkCompleteEvent<br>Set PhysLinkCompleteStatus to 0x00 (no error) | STARTING |
| HCI_Create_Physical_Link command | MAC already in selected channel | Determine selected channel<br>Send HCI Channel Select event<br>Set Connection Accept Timeout<br>Set NeedPhysLinkCompleteEvent<br>Set PhysLinkCompleteStatus to 0x00 (no error) | CONNECTING |
| HCI_Create_Physical_Link command | No suitable channel | Determine selected channel<br>Send HCI Physical Link Complete event with Status set to Connection Rejected due to Limited Resources (0x0D) | DISCONNECTED |
| HCI_Accept_Physical_Link command | MAC not yet in channel | Signal MAC to start on channel<br>Set Connection Accept Timeout<br>Set NeedPhysLinkCompleteEvent<br>Set PhysLinkCompleteStatus to 0x00 (no error) | STARTING |
| HCI_Accept_Physical_Link command | MAC already in that channel | Set Connection Accept Timeout<br>Issue MAC connection command<br>Set NeedPhysLinkCompleteEvent<br>Set PhysLinkCompleteStatus to 0x00 (no error) | CONNECTING |
| HCI_Accept_Physical_Link command | No suitable channel | Send HCI Physical Link Complete event with Status set to Connection Rejected due to Limited Resources (0x0D) | DISCONNECTED |

*Table 3.1: DISCONNECTED State event table*

### 3.1.8 STARTING State

This state is used to begin MAC operation, if required.

| Event | Condition | Action | Next State |
|---|---|---|---|
| Connection Accept Timeout | | SetPhysLinkCompleteStatus to Connection Accept Timeout (0x10) | DISCONNECTED |
| HCI_Disconnect_Physical_Link command | | Indicate HCI Disconnection Physical Link Complete event with Status set to Success (0x00) and Reason set to Connection Terminated By Local Host (0x16)<br>Clear Connection Accept Timeout<br>Cancel MAC connect operation<br>Set PhysLinkCompleteStatus to Unknown connection identifier (0x02) | DISCONNECTED |
| MAC Start Completed | Device is link originator | Issue HCI Channel Select event | CONNECTING |
| MAC Start Completed | Device is link responder | Issue MAC connection command | CONNECTING |
| MAC Start Failed | | Clear Connection Accept Timeout<br>Set PhysLinkCompleteStatus to MACConnection Failed(0x3F) | DISCONNECTED |

*Table 3.2: STARTING State event table*

### 3.1.9 CONNECTING State

This state is used to cause the devices to start communication with each other, over the 802.11 media.

| Event | Condition | Action | Next State |
|---|---|---|---|
| Connection Accept Timeout | | Cancel MAC connect operation<br>Set PhysLinkCompleteStatus to 0x10 (Connection Accept timeout) | DISCONNECTING |
| MAC Connect Completed | Device is responder | | AUTHENTICATING |

*Table 3.3: CONNECTING State event table*

| Event | Condition | Action | Next State |
|---|---|---|---|
| MAC Connect Completed | Device is initiator | Initiate four way handshake | AUTHENTICATING |
| HCI_ Disconnect_ Physical Link command | | Indicate HCI Disconnection Physical Link Complete event with Status set to Success (0x00) and Reason set to Connection Terminated By Local Host (0x16)<br>Set PhysLinkCompleteStatus to Unknown connection identifier (0x02)<br>Clear Connection Accept Timeout<br>Cancel MAC connect operation | DISCONNECTING |
| MAC Connect Failed | | Cancel MAC connect operation<br>Set PhysLinkCompleteStatus to MAC Connection Failed (0x3F) | DISCONNECTING |

*Table 3.3: CONNECTING State event table*

### 3.1.10 AUTHENTICATING State

The AUTHENTICATING state is entered when the two devices have established an unsecured connection.

While in the AUTHENTICATING state the two devices shall perform the 802.11 RSN 4-way handshake as described in Section 3.5, establish a PTKSA, and insert key material into the MAC.

| Event | Condition | Action | Next State |
|---|---|---|---|
| Connection Accept Timeout | | Set PhysicalLinkCompleteStatus to 0x10 (MAC Connection Failed) | DISCONNECTING |
| HCI_ Disconnect_Physical Link command | | Indicate HCI Disconnection Physical Link Complete event with Status set to Success (0x00) and Reason set to Connection Terminated By Local Host (0x16)<br>Set PhysLinkCompleteStatus to Unknown connection identifier (0x02<br>Clear Connection Accept Timeout<br>Signal MAC to disconnect peer | DISCONNECTING |

*Table 3.4: AUTHENTICATING State event table*

| Event | Condition | Action | Next State |
|---|---|---|---|
| 4-way Hand-shake Failed | | Signal MAC to disconnect peer<br>Set PhysLinkCompleteStatus to 0x05 (Authentication failure)<br>Clear Connection Accept Time-out | DISCONNECTING |
| 4-way Hand-shake Suc-ceeded | | Send HCI Physical Link Com-plete event with Status set to Success (0x00)<br>Configure MAC with Link Supervision Timeout<br>Clear Connection Accept Time-out<br>Clear NeedPhysLink CompleteEvent | CONNECTED |

*Table 3.4: AUTHENTICATING State event table*

### 3.1.11 CONNECTED State

The CONNECTED state is the operational state for the physical link.

| Event | Condition | Action | Next State |
|---|---|---|---|
| HCI_ Disconnect_ Physical Link command | | Indicate HCI Disconnection Physical Link Complete event with Status set to Success (0x00) and Reason set to Con-nection Terminated By Local Host (0x16)<br>Clear Connection Accept Time-out<br>Signal MAC to disconnect peer | DISCONNECTING |
| MAC Media Dis-connection Indi-cation | | Indicate HCI Disconnection Logical Link Complete event for each logical link, with Status set to Success (0x00) and Reason set to Connection Terminated by remote host<br>Cancel MAC connect operation | DISCONNECTING |

*Table 3.5: CONNECTED State event table*

### 3.1.12 DISCONNECTING State

This is a transit state to the DISCONNECTED state and is used to prevent the PAL from preparing to accept or create new connections while a given connec-tion is being deleted by the PAL and the MAC. The PAL shall exit from this state when the MAC is ready to accept new connections.

| Event | Condition | Action | Next State |
|---|---|---|---|
| MAC Connection Cancel Indication | PhysLinkCompleteEvent is set | Send HCI Physical Link Complete with status set to PhysLinkCompleteStatus | DISCONNECTED |
| MAC Connection Cancel Indication | PhysLinkCompleteEvent is clear | | DISCONNECTED |

*Table 3.6: DISCONNECTING State event table*

## 3.2 CHANNEL SELECTION

### 3.2.1 Overview

Peer to peer networks in 802.11 can incur long connection latency unless there is out of band coordination. Also, one or both of the AMP devices may already be connected to an external network and may therefore need to remain on a given channel. To solve these problems, the data structures and algorithms specified here may be used to provide a coordination service during the creation of the physical link.

An 802.11 channel should be chosen based on up-to-date dynamic information obtained from both AMP peers. The channel selection process is outside the scope of this document but the selected channel shall meet the following criteria:

- Shall be legally permitted for use according to local regulatory agencies. If no locale is known, then a common mode configuration shall be used. See Section 3.2.2.
- Shall not use a 40 MHz channel width in the 2.4 GHz ISM band.
- Shall be present in the Preferred Channel List TLV in the AMP_ASSOC given in the HCI Write Remote AMP ASSOC command.
- Should avoid forcing either AMP device to move its channel.
- Should favor channels with least impact on BR/EDR operation.

The initiating PAL may not be able to select a suitable channel or the responding PAL may reject the selected channel. If the selection process cannot identify a channel, then the physical link establishment shall be aborted.

The selected channel is transmitted to the responding AMP manager as a field in the AMP_ASSOC parameter in the AMP Create Physical Link message, and given to the responding PAL via the HCI Write Remote AMP ASSOC command.

### 3.2.2 Regulatory

Even though IEEE 802.11 devices operate in unlicensed spectral bands, the unlicensed bands and the allowable behavior in each band is specific to a country and is enforced by spectrum regulatory bodies. See annex I of [1] for example regulations and references to relevant documents. There is a consistent set of rules for operation in the 2.4 GHz ISM band, as described in the following text.

802.11 AMP equipment providers interpret these rules in independent ways and meet the regulations with independent mechanisms so specification of an algorithm to implement regulatory compliance for all situations is beyond the scope of this document.

If an implementation has knowledge of local regulatory constraints then such an implementation may be able to improve certain characteristics of the AMP link, for example by selecting a 5 GHz channel of operation to provide better performance with the collocated BR/EDR radio.

### 3.2.3 Specification of Channel Identifiers

The format of Preferred Channel Lists and Connected Channel Lists is specified by the 802.11 Country information element in [1] clause 7.3.2.9. Tables including channel lists and required behaviors are given for three regulatory domains in [1] normative annex J. From this it is seen that to resolve all ambiguity of channel identification, the country string as specified in [4] and the regulatory class are needed. Multiple regulatory triplets may be given in the same Country information element. If sub-band triplets are given then they are to be assumed to modify the immediately prior regulatory triplet. Sub-band triplets may be specified in any order, but they must not contain overlapping channel sets, as noted in [1] clause 7.3.2.9.

The Preferred Channel List shall be included in messages containing AMP_ASSOC fields exchanged over the BR/EDR link during construction of the physical link. The Preferred Channel List shall contain exactly one country string and one or more regulatory triplets. If the current locale is unknown, then the non-country designator of 'XXX' (see definition of dot11CountryString in [1] normative Annex D) and regulatory class of 254 shall be used. The set of channels preferred when using this designator and regulatory class shall be channels 1 through 11 in the 2.4 GHz ISM band, with channel 1 the most preferred of the set. Supporting examples follow.

In the first example preferred channel list, we assume the PAL does not know its locale and it prefers operation on channel 1 in the 2.4 GHz ISM band. The absence of a specific sub-band triplet implies all channels 1 through 11 are acceptable.

In the next example, assume the PAL again does not know its locale, but it prefers operation only on channels 6, 7 and 11, with 11 the most preferred.

| Field | Value | Comments |
|---|---|---|
| Country String | 'XXX' | Applies to entire table. Signifies non-country (mobile applications). |
| Regulatory Extension Identifier | 201 | |
| Regulatory Class | 254 | In context of Country String. Signifies 2.4 GHz ISM band channels 1 through 11 unless modified by specific channel list. |
| Coverage Class | 0 | Not used by AMPs. |

*Table 3.7: Simple preferred channel list*

| Field | Value | Comments |
|---|---|---|
| Country String | 'XXX' | Applies to entire table. Signifies non-country (mobile applications) |
| Regulatory Extension Identifier | 201 | |
| Regulatory Class | 254 | In context of Country String. Signifies 2.4 GHz ISM band channels 1 through 11 unless modified by specific channel list. |
| Coverage Class | 0 | Not used by AMPs. |
| First Channel | 11 | First sub-band triplet |
| Number of channels | 1 | |
| Maximum transmit power level | 20 | |
| First Channel | 6 | Second sub-band triplet |
| Number of channels | 2 | |
| Maximum transmit power level | 20 | |

*Table 3.8: Preferred channel list with non-contiguous channels*

In the final example, assume the PAL knows that it is operating in the US and it has determined that all channels in the 2.4 GHz ISM band and 2 channels in the 5 GHz U-NII I band should be communicated to the other peer. It is also assumed that the PAL has determined that channels in the 2.4 GHz band are preferred to the 5 GHz channels. The PAL would use the following Preferred Channel List.

| Field | Value | Comments |
|---|---|---|
| Country String | 'US' | Applies to entire table. |

*Table 3.9: Mixed band preferred channel list example*

| Field | Value | Comments |
|---|---|---|
| Regulatory Extension Identifier | 201 | First regulatory triplet. |
| Regulatory Class | 12 | In context of Country String, specifies channels 1 through 11, inclusive, in 2.4 GHZ ISM band. Absence of specific channel list signifies all channels in this (FCC) regulatory class are available to the AMP. |
| Coverage Class | 0 | Not used by AMPs. |
| Regulatory Extension Identifier | 201 | Second regulatory triplet. |
| Regulatory Class | 1 | In context of Country String, specifies 20 MHz channel spacing. Presence of subsequent channel list signifies not all channels in the regulatory class are available to the AMP. |
| Coverage Class | 0 | Not used by AMPS. |
| First channel | 36 | First sub-band triplet. |
| Number of channels | 2 | Channels 36 and 40 are included, with channel 36 more preferred than channel 40. |
| Maximum transmit power level | 20 | Units in context of Country String (dBm, mW, etc). |

*Table 3.9: Mixed band preferred channel list example*

## 3.3 802.11 LINK CREATION

### 3.3.1 Starting the AMP Network

After the initiator has selected a channel of operation, the initiating PAL shall instruct the MAC to commence network operation, including beaconing. The 802.11 AMP devices shall use probe responses and/or beacons (respecting regulatory restrictions) to advertise MAC capabilities.

AMP devices shall use beacons to enable effective coexistence with neighboring 802.11 networks. For example, non-AMP devices can discover the existence of AMP networks and such devices can also determine the EDCA characteristics of the network. In regulatory domains where DFS operation is required, an AMP device may need to passively observe the channel to check for radar, as described in [1] clause 11.9, before it may start to beacon. For AMP operation, the maximum beacon period shall be Max80211BeaconPeriod.

The SSID information element for AMP devices shall be of the form 'AMP-xx-xx-xx-xx-xx-xx' (with no null termination and no quotes) where the "x" charac-

ters are replaced by the lowercase hexadecimal characters of the MAC address of the local 802.11 device. This is referred to here as the AMP SSID. For example, if the MAC address of a device is 00:01:02:0A:0B:0C, then the AMP SSID would be 'AMP-00-01-02-0a-0b-0c'.

AMP beacons shall be indicated as ESS-style beacons in the capability information field as described in [1] clause 7.3.1.4 and shall include the AMP SSID. Beacons shall be sent with standard beacon channel access semantics as given in [1] clause 11.1.2.1. The contents of the Address2 and Address3 fields for beacons and probe responses shall be the MAC address of the transmitting AMP node. Probe requests sent to AMP peers shall use the MAC address of the intended recipient as the content of the Address1 and Address3 fields. In [1] clause 7.2.3.1 contains the details of the format of 802.11 beacon frames and [1] clause 7.2.3.9 contains the details of the format of 802.11 probe response frames, including a list of the required information elements is specified.

### 3.3.2 Establishing the 802.11 Link

AMP devices may choose to obtain the timebase of their peer through the timestamp field in 802.11 beacons and probe responses. The respective Target Beacon Transmission Time (TBTT) of the peers may be independent and occur at different times with respect to one another. The beacon period of the devices may be different from one another.

AMPs shall use RSN security. RSN security requires the use of 802.11 open authentication as specified in [1] clause 8.2.2.2. The AMP responder shall send the first frame of the 802.11 authentication transaction sequence with transaction ID of 1. Address fields Address1 and Address3 shall contain the initiator's address.The AMP initiator shall respond with an 802.11 authentication frame with transaction ID of 2. Address fields Address2 and Address3 shall contain the initiator's address.

AMPs shall use 802.11 (re)association frames to select features advertised by their peer. The AMP responder shall send an (re)association request frame. Address fields Address1 and Address3 shall contain the initiator's address.

The AMP initiator shall reply with an (re)association response frame. Address fields Address2 and Address3 shall contain the initiator's address. The frame body contents of both the (re)association request and response are given in [1] clause 7.2.3.4 and [1] clause 7.2.3.5, respectively.

After successful 802.11 (re)association, unencrypted security frames may be transmitted on the physical link.

### 3.3.3 Address Fields of Data Frames

The 802.11 AMP shall support the use of four address field frame format for all data frames after (re)association.

To use data frames with four address fields the AMP shall set the ToDS and FromDS bits in the FrameControl field equal to one. The addresses used for such frames shall be arranged as shown in Table 3.10. Because there is no frame forwarding by AMPs, the RA is the same as the DA and the TA is the same as the SA.

| Field | Value |
|---|---|
| Address1 | Receiver Address (RA) |
| Address2 | Transmitter Address (TA) |
| Address3 | Receiver Address (RA) |
| Address4 | Transmitter Address (TA) |

*Table 3.10: Four address frame address fields*

### 3.3.4 Admission Control

AMP devices should respond to probe requests with a probe response.

If Address2 of an 802.11 association request does not match the MAC address from the AMP Assoc received during construction of the current physical link or if the SSID in the association request does not match the AMP SSID of the receiving AMP device, then the receiving AMP device shall not transmit an 802.11 association response with a status code of 0 (success).

## 3.4 PHYSICAL LINK MAINTENANCE

After a physical link is created, an AMP device shall monitor the state of the link and provide an indication of link failure to the Host if no frames are received from the AMP physical link peer for a period of Link Supervision Timeout (LSTO). Correctly decrypted data frames received from the peer shall be evidence of an existing physical link, but this is not true for 802.11 ACK and CTS control frames.

If the PAL has not received a correctly decrypted data frame for a period less than LSTO, then it shall solicit a response from its peer in an attempt to receive the response before the expiration of LSTO. For this purpose, link supervision request/response protocol identifiers are given in Table 5.2 and may be used to construct link supervision data frames. When a PAL receives a data frame with a protocol identifier of Link Supervision Request, it shall reply by transmitting a data frame with a protocol identifier of Link Supervision Reply.

## 3.5 PHYSICAL LINK SECURITY

### 3.5.1 Obtaining Key Material

The Host provides key material for use with a physical link as the Dedicated_AMP_Link_Key parameter of the HCI_Create_Physical_Link or HCI_Accept_Physical_Link command. See [Vol 2] Part E, Section 5.2 for octet ordering of multi-octet HCI parameter values. The Dedicated_AMP_Link_Key parameter shall be interpreted by the PAL as a 256 bit integer and used directly as a Pairwise Master Key (PMK) by the two devices to create a PTK. Further key material is derived from the PTK.

### 3.5.2 Creating a PTK

The 802.11 4-way handshake is used to create a PTK from the PMK. See [1] clause 8.5.3.

Entities known as authenticator and supplicant are used to exchange security information in the 802.11 security architecture. Since the responding AMP device receives the 802.11 (re)association response frame, it shall serve the role of supplicant; the initiating AMP device shall serve the role of authenticator. The authenticator sends the first and third messages of the 4-way handshake and the supplicant sends the second and fourth messages. Only a single instance of the 4-way handshake is run by the 802.11 AMP to establish its secure connection.

If a 4-way handshake fails then the PAL physical link state machine shall transition to the DISCONNECTING state as described in Section 3.1.10.

The supported security configurations of the peers are given in the RSN information element in beacons or probe responses exchanged by the AMP devices. The PAL shall enforce the following restrictions:

- UseGroup (00:0F:AC:00), WEP-40 (00:0F:AC:01), TKIP (00:0F:AC:02), and WEP-104 (00:0F:AC:05) shall not be allowed as valid pairwise cipher suites
- The only valid AKMP shall be PSK (00:0F:AC:02) or a vendor-specific AKMP
- The NoPairwise bit (B1) of the RSN Capabilities field shall not be allowed as a valid selection
- The group cipher shall be CCMP (00:0F:AC:04)

The supplicant shall ensure a proper intersection of capabilities exists subject to the constraints above. It may also choose a set according to its policy requirements and may decide to terminate a connection attempt if no policy match is found. Either the supplicant or the authenticator may choose to terminate a connection after AMP Create Physical Link Response is sent by indicating an HCI Physical Link Complete event with an unsuccessful status code.

The AMP key received from the host may be marked with a Link_Type of debug. If it is, then the local PAL may choose to use the key, or it may choose to refuse to establish the link, according to its own policy. The management of this debug key policy is outside the scope of this document.

The PAL shall use the AMP key as a Pairwise Master Key (PMK) according to the 802.11 key hierarchy described in [1] clause 8.5.1.2 and shall exchange nonces (as part of a 4-way handshake) to add liveness in the derivation of a Pairwise Transient Key (PTK). The PTK shall be derived specifically for a session started by the HCI_Create_Physical_Link_Request command and shall be destroyed when the session is terminated by the HCI_Disconnect_Physical_Link command. The construction of the AAD used with CCMP shall include Address4 in the manner illustrated in [1] clause 8.3.3.3.2, Figure 8-17.

### 3.5.3 Using Encryption

Encryption keys are derived from the PTK and are inserted into the 802.11 MAC after the 4-way handshake has successfully completed. All data frames after this point are encrypted and this state persists until the physical link is destroyed.

### 3.5.4 Refreshing a PTK

The PTKSA, if any, shall be discarded when the physical link to which it applies enters the DISCONNECTED state. At this point the Host may reestablish the connection which will cause the creation of a new PTKSA.

The PTKSA shall be discarded in the event its receive sequence counter becomes exhausted. Since this is a 48 bit counter, this is an unlikely event.

To establish a new PTK, the physical link shall be torn down and re-established.

### 3.5.5 Transporting Security Handshake Messages

Security handshake messages shall be sent after the physical link is created, but before any logical link is created between the two devices.

The SNAP header composed of the OUI of the Bluetooth SIG and the protocol identifier given in Table 5.2 shall be used to distinguish AMP 4-way handshake messages from external security traffic.

## 3.6 PHYSICAL LINK SUPPORT FOR QOS

### 3.6.1 QoS Advertisement

If an AMP device supports 802.11 EDCA and is configured to use it, then it should indicate this to the host by setting the Guaranteed_Service_Type_Supported field of the PAL_Capabilities parameter included in the HCI_Read_Local_AMP_Info command.

If QoS is offered by an AMP device, it shall advertise EDCA in beacon and probe response frames by including the EDCA Parameter Set information element, as given in [1] clause 7.3.2.29. Note: The EDCA Parameter Set information element is also included in (re)association response frames. For AMP devices, the use of the EDCA parameter set shall be as follows. The QoS Info field shall be zero. The ACI, AIFSN, and TXOPlimit values of the AC parameter record shall be as given in [1] Table 7-37. The ECWmin and ECWmax values are specific to the 802.11 PHY type and are documented in their appropriate clauses in [1] The Admission Control Mandatory (ACM) bit should be zero.

The QoS Capability information element is described in [1] clause 7.3.2.35. The QoS Info field (see [1] clause 7.3.1.17) is contained in the QoS Capability information element as well as the first field of the EDCA Parameter Set information element. AMP devices shall not include the QoS Capability information element in beacons or probe responses.

The content of the QoS Info field is different depending on if the enclosing frame is a beacon or probe response, or if it is a (re)association request. The QoS Info field is not included in (re)association response frames.

| 802.11 Information element | Frame(s) contained in |
|---|---|
| QoS Capability information element | Association Request, Reassociation Request |
| EDCA Parameter Set information element | Beacon, Probe Response, Association Response, Reassociation Response |

*Table 3.11: EDCA advertisement and negotiation*

The EDCA AC parameters shall not change unless the physical link is torn down and re-established. Therefore the EDCA Parameter Set Update Count subfield of the QoS Info field in beacons and probe responses shall be zero.

### 3.6.2 Negotiation

To request the use of EDCA on the physical link an AMP device shall include a QoS Capability element in its (re)association request. If the AMP peer rejects the EDCA negotiation then the (re)association response frame shall have no EDCA Parameter Set element included. If the (re)association response frame has a status code of successful and the EDCA Parameter Set element is included, then the link shall be considered to support EDCA.

# 4 LOGICAL LINK MANAGER

A logical link provides a (possibly) bidirectional path for in-order delivery of L2CAP PDUs, with a specified set of traffic characteristics. Each logical link exists with respect to a specific physical link in the CONNECTED state.

A logical link is characterized by a pair of Extended Flow specification parameter sets, as described in [1]. An Extended Flow specification parameter set is referred to simply as a flow spec here.

Each logical link is classified as either Best Effort or Guaranteed. The logical link is known as Best Effort if either of its flow specs indicates Best Effort in its Service Type field. Otherwise, it is known as Guaranteed.

Support for Guaranteed links is optional; the PAL may reject any guaranteed flow spec. If 802.11 QoS (see Section 3.6.1) is not supported by both endpoints, then there is no traffic prioritization in the MAC.

## 4.1 LOGICAL LINK CREATION

Creation of a logical link is initiated by the HCI_Create_Logical_Link or HCI_Accept_Logical_Link command. If the physical link is not in state CONNECTED, the PAL shall send the HCI Logical Link complete event with status set to Command Disallowed (0x0C). The Controller shall indicate successful completion of the logical link creation using the HCI Logical Link Complete event, with Status set to Success (0x00).

### 4.1.1 Logical Link Handles

When a Logical Link is created or accepted the PAL shall create a Logical Link handle, or logical handle. The logical handle is included in HCI ACL data packets received from the HCI in the Handle field; the PAL may use the logical handle to help it select the egress physical link. If the Host delivers an HCI ACL data packet to the AMP Controller with an invalid Handle field, then the AMP shall discard the ACL data packet and the PAL may indicate a Number of Completed Blocks or Number of Completed Packets event (depending on the configuration of the Flow Control setting) using the Host-supplied logical handle.

When the PAL receives data frames from the MAC, it may use the source address to determine the physical link. It shall place the physical link handle corresponding to that link into the Handle field of the HCI ACL data packet before the packet is indicated to HCI. Note: the PAL is not required to determine a logical handle for frames it receives from the MAC.

### 4.1.2 Null Traffic Logical Links

Data frames may be discarded if attempted on a No Traffic flow.

### 4.1.3 Best Effort Logical Links

There is a single logical link used to transport all Best Effort traffic.

If EDCA was successfully negotiated during the physical link creation, the PAL shall map all egress frames marked with the Best Effort logical handle to a UP of either BEUserPrio0 or BEUserPrio1, inclusive. Otherwise, if EDCA is not used, no mapping is required and the UP shall be set to zero by the PAL.

### 4.1.4 Guaranteed Logical Links

Flow specs requesting guaranteed service may be accepted by the PAL if the physical link was negotiated with 802.11 QoS.

EDCA shall be used by the 802.11 AMP devices if supported and available for use by both devices. The PAL shall provide a priority field associated with each egress 802.11 frame. The priority field is interpreted by the MAC as a User Priority (UP) and is mapped to access category as specified in [1] clause 9.1.3.1.

If the 802.11 link was negotiated with QoS, then on receiving an HCI_Create_Logical_Link or HCI_Accept_Logical_Link command specifying a Guaranteed transmit flowspec the PAL shall use a UP with a value between MinGUserPrio and MaxGUserPrio, inclusive. The determination of precisely which UP to use is outside of the scope of this specification. A flow spec expresses maximum bandwidth as the product of inter-SDU arrival time and maximum SDU size. The PAL may use the specified maximum bandwidth or latency requirements from the flow spec to establish mappings of logical handles to UPs.

If a request for a guaranteed link cannot be mapped to a UP in the range specified above, it may be rejected.

The PAL shall reject all requests to establish a guaranteed logical link with a flow spec expressing a maximum bandwidth which is greater than the Total_Bandwidth parameter of the Read Local AMP Info command minus the sum of the requested maximum bandwidths of all existing guaranteed logical links.

The transmitter shall store the logical channel to UP mapping for application to future frame transmissions. The receiver shall create an HCI ACL data header and insert the physical link handle into the Handle field of the packet before indication to the Host.

## 4.2 LOGICAL LINK UPDATES

The Host may indicate a change of traffic requirements on a logical link by using the HCI_Flow_Spec_Modify command. An HCI_Flow_Spec_Modify command will not change the Service Type of the flow specs for a logical link.

When the PAL receives an HCI_Flow_Spec_Modify command, it should validate that the new flow spec requirements can be met with the available resources. If not, then the PAL should reject the command.

## 4.3 LOGICAL LINK DELETION

The PAL shall rely on upper layers to flush any frames before a logical link is destroyed. See explicit flush in Section 5.3. The PAL may choose to re-use the same logical link identifier for new logical links even on the same physical link.

The PAL should recover any allocated QoS resources when the logical link is deleted. In particular, when deallocating resources for a flow spec which expressed a maximum bandwidth, the PAL should subtract that parameter from the total allocated bandwidth.

# 5 DATA MANAGER

## 5.1 ENCAPSULATION

The PAL shall advertise a maximum PDU length of Max80211PALPDUSize octets and each L2CAP PDU is transmitted by the MAC as a single MSDU. The MSDU boundary determines the L2CAP PDU boundary for the receiver.

Before transmission, the PAL shall remove the HCI header, add LLC and SNAP headers and insert an 802.11 MAC header. The LLC/SNAP frame format used by the 802.11 AMP is shown in Table 5.1.

| | DSAP | SSAP | Control | OUI | Protocol | Frame Body |
|---|---|---|---|---|---|---|
| Value | 0xAA | 0xAA | 0x03 | 00:19:58 | XX:XX | |
| Octets | 1 | 1 | 1 | 3 | 2 | 0-1492 |

*Table 5.1: 802.11 AMP LLC/SNAP encapsulation*

The protocol identifiers shall be as shown in Table 5.2:

| Value | Protocol Description | Logical Link |
|---|---|---|
| 0x0000 | Reserved | N/A |
| 0x0001 | L2CAP ACL data | AMP-U |
| 0x0002 | Activity Report | AMP-C |
| 0x0003 | Security frames | AMP-C |
| 0x0004 | Link supervision request | AMP-C |
| 0x0005 | Link supervision reply | AMP-C |
| 0x0006-0xFFFF | Reserved | N/A |

*Table 5.2: Protocol Identifiers*

All 802.11 data frames on the AMP link shall be sent with ToDS and FromDS bits in the Frame Control field both set to one. For a description of the Frame Control field, see [1] clause 7.1.3.1. If QoS was negotiated on the physical link between the peers then the QoS Control field shall be included in the MAC header, otherwise it shall not be included.

The receiving device can determine the physical link identity from the TA of the received frame. The receiving PAL shall decapsulate the frame from 802.11 and into the HCI ACL data encapsulation.

## 5.2 COEXISTENCE AND LOCAL INTERFERENCE

### 5.2.1 Interference from Collocated Radios

The BR/EDR radio, LE PHY and the ERP of 802.11 AMP operate in the 2.4GHz ISM band and mechanisms are required to help mitigate potential interference. The BR/EDR and LE radio subsystems on an 802.11 AMP capable device should employ AFH to attempt to avoid interference of overlapping transmissions on the medium.

Protocols specified in [1] are designed to mitigate interface from non-collocated radios.

On systems where the devices are collocated such that radio isolation is insufficient to mitigate interference, the use of the shared medium should be time-division multiplexed to ensure only one of the interfering radios will gain access to the medium. In the case of the BR/EDR radio and the 802.11 radio operating in the 2.4GHz band, the PAL should ensure that local high priority BR/EDR traffic such as SCO, eSCO and ACL packets carrying A2DP information have higher priority over potentially interfering local 802.11 AMP packets. The PAL may ensure this prioritization through many ways including employing the methods described in [3] but the exact methods are outside the scope of this document.

Interference can also arise between the 802.11 AMP radio and collocated licensed band radios (LBRs) operating in adjacent bands to the ISM spectrum. Due to 802.11 AMP transmissions, the collocated LBR may not be able to receive transmissions from its peer LBR. Again the use of the medium should be time-division multiplexed to ensure only one of the radios will gain access to the medium at one time. In the case of the 802.11 AMP radio and an LBR, the PAL should ensure that the local LBR packets have higher priority over potentially interfering local 802.11 AMP packets. Although the exact methods are outside the scope of this document, similar collocated radio interference mitigation mechanisms as described above may be used.

### 5.2.2 Unavailability of Remote Peer

Distributed AMP devices may experience 802.11 performance degradation due to simultaneous BR/EDR activity occurring at one of the AMP peers. Local interference mitigation schemes (see Section 5.2.1) employing time-divided access to the medium will result in the 802.11 radio being periodically unavailable to its peer. The PAL attempting to transmit to a periodically unavailable AMP device may employ techniques to allow its transmissions to consume minimal airtime and power while still achieving acceptable performance for its own transmissions and those of neighboring networks.

When a physical link is created, the PAL shall configure the 802.11 MAC to use RTS/CTS signaling by default. This behavior may be modified by using Activity Reporting as given in Section 5.2.3.

### 5.2.3 Activity Reports

AMP Activity Reports provide an optional mechanism for the 802.11 PAL to inform its 802.11 AMP peer of events which may result in the 802.11 device being unavailable to receive 802.11 traffic. Activity Reports may also be used to inform a peer of the absence of local interference thereby allowing the remote peer to disable its RTS/CTS signaling.

Examples of simultaneous traffic include, but are not limited to:

1. BR/EDR SCO/eSCO streams
2. 802.11 traffic required to maintain an external 802.11 connection
3. Traffic from other collocated radios such as LBRs in the 2.3 or 2.5 GHz band (including WiMax, LTE, and UMB)

When Activity Reports are used, it is the PAL which is aware of its unavailability which may generate the interference information. The PAL can determine the information to include in the Activity Report through methods including, for example, the PTA model described in [3]. In many systems with collocated radios there are PTA signals between the BR/EDR and 802.11 controllers which may be used to generate this information.

The information shall be transferred between the communicating 802.11 PALs using a PAL Activity Report PDU encapsulated in an 802.11 data frame. The frame body of the Activity Report PDU following the LLC/SNAP header is shown in Table 5.3. It has a variable length.

*Activity Report* *Size: Variable*

| Value | Octets | Description |
|---|---|---|
| ScheduleKnown | 1 | Bit 0:<br>1 if the sender knows the schedule of interference<br>0 if the sender does not know the schedule of interference |
| NumReports | 1 | The number of traffic reports in this PDU |
| StartTime | 4 | The absolute time of the start of possible unavailability of the peer, expressed as the least significant 32 bits of the 802.11 TSF of the transmitter of the PDU. |
| Duration | 4 | Duration of the active phase of the traffic in microseconds. |
| Periodicity | 4 | Periodicity of traffic in microseconds. May be zero to indicate aperiodic traffic. |

*Table 5.3: Activity Report*

Processing of received Activity Reports is optional and support for it is advertised in the 802.11 PAL Capabilities field.

The ScheduleKnown field shall indicate to the receiver of the Activity Report whether or not the sender is aware of the schedule of subsequent data traffic.

- If the ScheduleKnown is set to zero, this shall signify the sender is aware of the presence of traffic but not its schedule, and the receiving PAL shall configure the MAC with RTS/CTS signaling for all traffic on the physical link to the sender.
- If the ScheduleKnown is set to one, the sender shall describe the schedule of interference in the subsequent fields; the receiver of the report may either schedule all 802.11 AMP traffic around this schedule, configure the MAC with RTS/CTS signaling, or both.

If a PAL knows there is no interference from collocated radios before or during establishment of a physical link, an Activity Report conveying this state should be sent at the earliest possible opportunity after the physical link is established.

The StartTime, Duration and Periodicity form an Activity Report triplet. The difference between the StartTime and the current TSF of the peer shall be interpreted as a signed 32 bit value. A negative value shall denote a time in the past. The NumReports value indicates the number of Activity Report triplets which follow. The Duration shall specify the amount of time (in microseconds) the PAL is in the mode specified. If the ScheduleKnown field is zero, there shall be no Activity Report triplets and the NumReports field shall be zero. If the Periodicity is non-zero, it shall be a value greater than the Duration.

An Activity Report may be created to describe a burst of interfering traffic by setting ScheduleKnown to one, NumReports to one, StartTime to the time the burst is predicted to start, Duration to the duration of the burst, and Periodicity to zero.

The PAL may inform its peer that interference is no longer present by sending an Activity Report with a ScheduleKnown field set to one, and a NumReports field set to zero. Upon reception of such an Activity Report frame, a receiving PAL may configure the MAC without RTS/CTS signaling.

The PAL may transmit an Activity Report frame periodically in order to correct clock drift between its TSF and the schedule of unavailability caused by the collocated radio. The most recent Activity Report shall supersede all others received.

## 5.3 EXPLICIT FLUSH

Explicit flush may be initiated at the HCI on any logical link by using the HCI_Enhanced_Flush command. Explicit flush discards all data frames for transmit on the indicated logical link.

## 5.4 AUTOMATIC FLUSH

If guaranteed logical links are supported, then automatic flush timeouts shall be supported by the 802.11 PAL.

## 5.5 QUALITY OF SERVICE VIOLATIONS

The PAL may generate a HCI QoS Violation event when it is determined that the parameters of the flow spec are not being met. This may occur for instance when a data packet has been queued for transmission on a Guaranteed logical link for longer than the Access Latency in the flow spec for transmitted traffic on that link. It can also occur when a data packet fails to receive an acknowledgement before the specified flush timeout.

# 6 CONSTANTS

| Name | Value | Units | Description |
|---|---|---|---|
| Max80211PALPDUSize | 1492 | Octets | Maximum PDU size |
| Max80211AMPASSOCLen | 672 | Octets | Maximum length of AMP_ASSOC for this AMP |
| MinGUserPrio | 4 | N/A | Minimum value of user priority for guaranteed link |
| MaxGUserPrio | 7 | N/A | Maximum value of user priority for guaranteed link |
| BEUserPrio0 | 0 | N/A | Best effort User Priority |
| BEUserPrio1 | 3 | N/A | Best effort User Priority |
| Max80211BeaconPeriod | 2000 | milliseconds | Maximum value of AMP dot11BeaconPeriod MIB variable |
| ShortRangeModePowerMax | 4 | dBm | Maximum transmit power for ERP-OFDM frames when in Short Range Mode |

*Table 6.1: 802.11 PAL Constants*

# 7 MESSAGE SEQUENCE CHARTS

The MSCs necessary to show the creation and deletion of physical and logical links can be found in the HCI specification (see [Vol 2] Part E), as the sequencing of steps is determined by the logical HCI. However, an overview MSC for physical link creation is given in Figure 7.1.

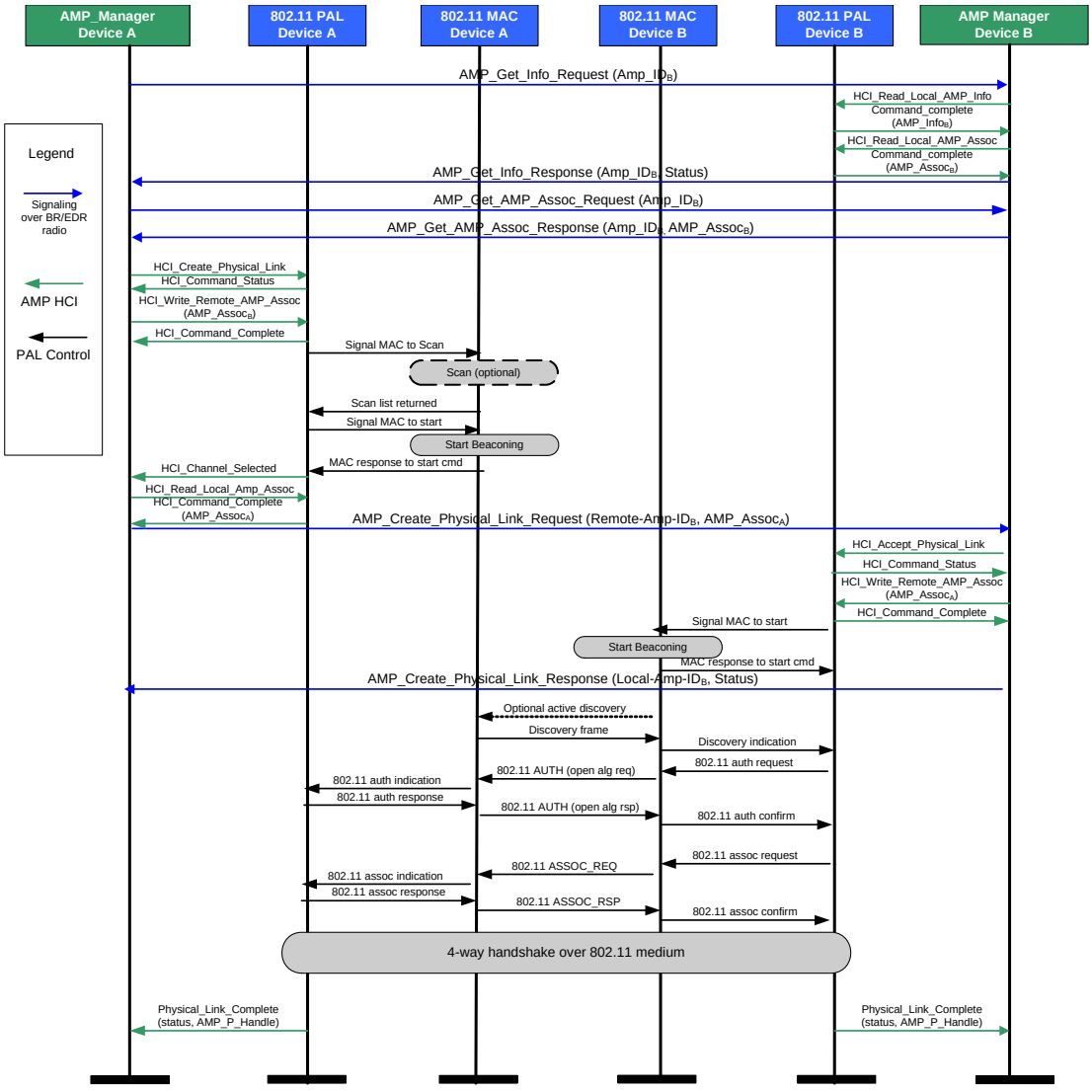



*Figure 7.1: Overview MSC for physical link create/accept*

# 8 APPENDIX A: TEST SUPPORT

This section provides the details of the AMP test commands and events described in [Vol 2] Part E.

## 8.1 AMP TEST COMMAND

| Command | OCF | Command Parameters | Return Parameters |
|---|---|---|---|
| HCI_AMP_Test | 0x0009 | Test Parameters | Status |

**Description:**

This command is used to configure and start a test. This command shall be only valid in AMP Test Mode.

When a test scenario has completed or on receiving a HCI_AMP Test_End command the AMP shall send a HCI AMP Test End event and the AMP returns to an idle state with the RX and TX off.

*Test Parameters:* *Size: 18 octets*

| Value | Parameter Description |
|---|---|
| 0xXX | Test Scenario<br>0x01 - Transmit single frames with following parameters<br>0x02 - Receive frames with following parameters<br>0x03 – 0xFF Reserved |
| 0xXX | Preamble<br>0x00 – ERP-OFDM preamble<br>0x01 - Short Preamble<br>0x02 - Long preamble<br>0x03 – 0xFF Reserved |
| 0xXX | Payload<br>0x00 – All Zeros payload<br>0x01 – All ones payload<br>0x02 - PRBS9. The PRBS9 sequence is reinitialized for every frame. Each PRBS9 payload is the same.<br>0x03 - PRBS15. The PRBS15 sequence is reinitialized for every frame. Each PRBS15 payload is the same.<br>0x04 – 0xFF Reserved |

| | | |
|---|---|---|
| 0xXXXXXX | Country | Channel Descriptor<br>For AMP type 802.11, the channel is completely described by a four-tuple of {Country, Regulatory Extension Identifier, Regulatory class, Channel number}. The Country identifier is an ISO/IEC-3166 three-octet field and the Regulatory Extension Identifier is equal to 201.<br>If the locale is unknown to the EUT, then it shall only allow channel numbers as given in Ref [2] Clause 18.4.6.2. Valid values are from 1 to 11.<br>All other values are reserved |
| 0xC9 | Regulatory Extension Identifier | |
| 0xXX | Regulatory Class | |
| 0xXX | Channel Number | |
| 0xXX | Modulation<br>0x00 ERP-DSSS<br>0x01 ERP-CCK<br>0x02 ERP-OFDM<br>0x03 ERP-PBCC<br>0x04 DSSS-OFDM<br>0x05 OFDM<br>All other values are reserved | |
| 0xXX | Rate (Mb/s)<br>Transmission data rate of PSDU.<br>See Ref [2] Clause 19.8.2 PHY MIB dot11SupportedDataratesTxValues<br>The allowed data rates are dependent on the modulation selected. Ref [2] Table 19.1 Clause 19.2 and Clause 17.2.3.3.<br>All other values reserved | |
| 0xXXXX | Payload length<br>1 to 1500. All other values reserved. | |
| 0xXX | Transmit Power Control (TPC)<br>Valid values are 1 to 8 as defined in the Ref [2] implementation dependent.<br>All other values reserved. | |
| 0xXX | Duty Cycle<br>10 to 99% (default 50%)<br>All other values reserved. | |
| 0xXXXX | Frame count<br>1 to 65525 - Number of frames to be transmitted. When the defined frame count has been transmitted the system returns to the idle state and the AMP Test End event is returned to the tester.<br>On receiving the Test End command the AMP returns to idle state. | |
| 0xXX | Scramble state<br>0x00 – OFF<br>0x01 – ON<br>All other values reserved. | |

**Return Parameters:**

*Status:* *Size: 1 Octet*

| Value | Parameter Description |
|---|---|
| 0x00 | AMP Test command succeeded |
| 0x01-0xFF | Test command failed. See "Error Codes" on page 339 [Part D]. |

**Event(s) Generated (unless masked away):**

When the AMP receives the HCI_AMP_Test command, the AMP shall send the Command_Status event to the AMP Test Manager which shall be routed to the tester.

The HCI AMP Start Test event shall be generated when the HCI_AMP_Test command has completed and the first data is ready to be sent or received.

The HCI Command Complete event shall not be sent by the AMP to indicate that this command has been completed. Instead the HCI AMP Start Test event shall indicate that this command has been completed.

When in a transmitter test scenario and the frames/bursts count have been transmitted the HCI AMP Test End event shall be sent.

### 8.1.1 Test Scenarios

All test mode frames shall be test mode data frames.

**Single Frame Transmission**

When the test scenario is set to transmit single frames the format shall be as defined by the parameters in the rest of the test configuration parameters. The interval between frames shall be as defined by the transmission interval time.

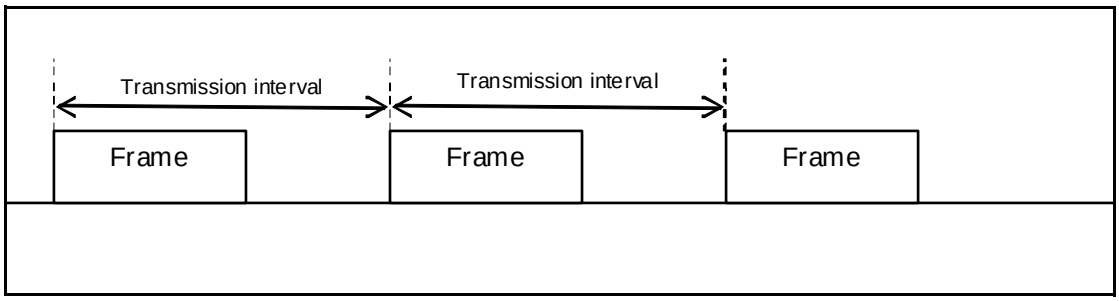



*Figure 8.1: Single Frame test transmission*

The transmissions shall continue until the frame count is reached or a test end command is received; when a HCI AMP Test End event shall be sent to the tester via the AMP Test Manager.

The requirements of the frames are described in the Test mode data frame format section.

#### Receive Frames

When the test scenario is set to receive frames the AMP shall receive the frames defined in the other test scenario parameters and, when AMP receiver reports are configured, return receiver reports as defined to the tester via the AMP Test Manager.

### 8.1.2 Test Mode Data Frame Format

The frames transmitted in test mode are not super frames. All frames shall be Data frames. The format of the frame agrees with the PHY frames in [2] section 19 with non connection test mode fixed fields.

#### Source and Destination Address

When performing the AMP PHY non connection tests these fields are fixed according to the direction of the message. The AMP shall use a fixed address AMP_TEST_ADDRESS and shall expect to receive frames with an AMP_TESTER_ADDRESS.

| Parameter | Address |
|---|---|
| AMP_TEST_ADDRESS | 0x5555 (0101010101010101 binary) |
| AMP_TESTER_ADDRESS | 0xAAAA (1010101010101010 binary) |

These are fixed addresses for testing purposes only.

The AMP shall transmit frames in the transmit test scenarios with the SrcAddr set to the AMP_TEST_ADDRESS and the DestAddr set to the AMP_TESTER_ADDRESS.

The AMP shall receive test frames in the receiver test scenarios with the SrcAddr set to AMP_TESTER_ADDRESS and the DestAddr set to the AMP_TEST_ADDRESS.

## 8.2 AMP START TEST EVENT

| Event | Event Code | Event Parameters |
|---|---|---|
| HCI_AMP_Start_Test | 0x49 | Status<br>Test Scenario |

#### Description:

The HCI AMP Start Test event shall be generated when the HCI_AMP_Test command has completed and the first data is ready to be sent or received.

**Event Parameters:**

*Status:* *Size: 1 Octet*

| Value | Parameter Description |
|---|---|
| 0x00 | Test command succeeded |
| 0x01-0xFF | Test command failed. See "Error Codes" on page 339 [Part D]. |

*Test Scenario:* *Size: 1 Octet*

| Value | Parameter Description |
|---|---|
| 0xXX | 0x01 - Transmit single<br>0x02 – Receive frames<br>0x00 and 0x03 – 0xFF Reserved |

## 8.3 AMP TEST END EVENT

| Event | Event Code | Event Parameters |
|---|---|---|
| HCI_AMP_Test_End | 0x4A | Status<br>Test Scenario |

**Description:**

The HCI AMP Test End event shall be generated to indicate that the AMP has transmitted or received the number of frames/bursts configured.

If the Receiver reports are enabled an HCI_AMP Receiver Report event shall be generated.

**Event Parameters:**

*Status:* *Size: 1 Octet*

| Value | Parameter Description |
|---|---|
| 0x00 | AMP Test command succeeded |
| 0x01-0xFF | Test command failed. See "Error Codes" on page 339 [Part D]. |

*Test Scenario:* *Size: 1 Octet*

Test Scenario for which the test end event has been generated.

| Value | Parameter Description |
|---|---|
| 0xXX | 0x01 - Transmit single<br>0x02 – Receive frames<br>0x03 – 0xFF Reserved |

# 9 REFERENCES

[1] IEEE 802.11-2007 Standard and Amendment 1 (Radio Resource Measurement)

[2] ISO/IEC 3166-2

[3] IEEE 802.15.2 Recommended Practice: Coexistence of WPAN with other wireless devices operating in unlicensed frequency bands

[4] Bluetooth SIG Company Identifiers https://www.bluetooth.org/Technical/AssignedNumbers/identifiers.htm

# 10 LIST OF FIGURES

Figure 1.1: Internal structure of the 802.11 PAL ..... 11
Figure 3.1: Physical Link finite state machine diagram ..... 24
Figure 7.1: Overview MSC for physical link create/accept ..... 49
Figure 8.1: Single Frame test transmission ..... 52

# 11 LIST OF TABLES

Table 2.1: TLV format....19
Table 2.2: TypeIDs used for 802.11 AMP TLVs....19
Table 3.1: DISCONNECTED State event table....27
Table 3.2: STARTING State event table....28
Table 3.3: CONNECTING State event table....28
Table 3.4: AUTHENTICATING State event table....29
Table 3.5: CONNECTED State event table....30
Table 3.6: DISCONNECTING State event table....31
Table 3.9: Mixed band preferred channel list example....33
Table 3.7: Simple preferred channel list....33
Table 3.8: Preferred channel list with non-contiguous channels....33
Table 3.10: Four address frame address fields....36
Table 3.11: EDCA advertisement and negotiation....39
Table 5.1: 802.11 AMP LLC/SNAP encapsulation....43
Table 5.2: Protocol Identifiers....43
Table 5.3: Activity Report....46
Table 6.1: 802.11 PAL Constants....48



Specification Volume 6

# SPECIFICATION OF THE *BLUETOOTH* SYSTEM

Experience More



# Core System Package [Low Energy Controller volume]

Covered Core Package version:
4.0

Bluetooth®



Ford Ex. 1052

## Revision History

The Revision History is shown in the [Vol 0] Part C, Section  on page 55.

## Contributors

The persons who contributed to this specification are listed in the [Vol 0] Part C, Section  on page 55.

## Web Site

This specification can also be found on the official Bluetooth web site:
http://www.bluetooth.com

## Disclaimer and Copyright Notice

The copyright in these specifications is owned by the Promoter Members of Bluetooth SIG, Inc. (“Bluetooth SIG”). Use of these specifications and any related intellectual property (collectively, the “Specification”), is governed by the Promoters Membership Agreement among the Promoter Members and Bluetooth SIG (the “Promoters Agreement”), certain membership agreements between Bluetooth SIG and its Adopter and Associate Members (the “Membership Agreements”) and the Bluetooth Specification Early Adopters Agreements (“1.2 Early Adopters Agreements”) among Early Adopter members of the unincorporated Bluetooth special interest group and the Promoter Members (the “Early Adopters Agreement”). Certain rights and obligations of the Promoter Members under the Early Adopters Agreements have been assigned to Bluetooth SIG by the Promoter Members.

Use of the Specification by anyone who is not a member of Bluetooth SIG or a party to an Early Adopters Agreement (each such person or party, a “Member”), is prohibited. The legal rights and obligations of each Member are governed by their applicable Membership Agreement, Early Adopters Agreement or Promoters Agreement. No license, express or implied, by estoppel or otherwise, to any intellectual property rights are granted herein.

Any use of the Specification not in compliance with the terms of the applicable Membership Agreement, Early Adopters Agreement or Promoters Agreement is prohibited and any such prohibited use may result in termination of the applicable Membership Agreement or Early Adopters Agreement and other liability permitted by the applicable agreement or by applicable law to Bluetooth SIG or any of its members for patent, copyright and/or trademark infringement.

THE SPECIFICATION IS PROVIDED “AS IS” WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, SATISFACTORY QUALITY, OR REASONABLE SKILL OR CARE, OR ANY WARRANTY ARISING OUT OF ANY COURSE OF DEALING, USAGE, TRADE PRACTICE, PROPOSAL, SPECIFICATION OR SAMPLE.

Each Member hereby acknowledges that products equipped with the Bluetooth® technology (“Bluetooth® Products”) may be subject to various regulatory controls under the laws and regulations of various governments worldwide. Such laws and regulatory controls may govern, among other things, the combination, operation, use, implementation and distribution of Bluetooth® Products. Examples of such laws and regulatory controls include, but are not limited to, airline regulatory controls, telecommunications regulations, technology transfer controls and health and safety regulations. Each Member is solely responsible for the compliance by their Bluetooth® Products with any such laws and regulations and for obtaining any and all required authorizations, permits, or licenses for their Bluetooth® Products related to such regulations within the applicable jurisdictions. Each Member acknowledges that nothing in the Specification provides any information or assistance in connection with securing such compliance, authorizations or licenses. NOTHING IN THE SPECIFICATION CREATES ANY WARRANTIES, EITHER EXPRESS OR IMPLIED, REGARDING SUCH LAWS OR REGULATIONS.

ALL LIABILITY, INCLUDING LIABILITY FOR INFRINGEMENT OF ANY INTELLECTUAL PROPERTY RIGHTS OR FOR NONCOMPLIANCE WITH LAWS, RELATING TO USE OF THE SPECIFICATION IS EXPRESSLY DISCLAIMED. BY USE OF THE SPECIFICATION, EACH MEMBER EXPRESSLY WAIVES ANY CLAIM AGAINST BLUETOOTH SIG AND ITS PROMOTER MEMBERS RELATED TO USE OF THE SPECIFICATION.

Bluetooth SIG reserves the right to adopt any changes or alterations to the Specification as it deems necessary or appropriate.

Copyright © 1999 - 2010

Ericsson AB,
Lenovo,
Intel Corporation,
Microsoft Corporation,
Motorola, Inc.,
Nokia Corporation,
Toshiba Corporation

*Third-party brands and names are the property of their respective owners.

# TABLE OF CONTENTS

**Part A**
**PHYSICAL LAYER SPECIFICATION**

**1 Scope ..... 14**
**2 Frequency Bands and Channel Arrangement ..... 16**
**3 Transmitter Characteristics ..... 17**
3.1 Modulation Characteristics ..... 17
3.2 Spurious Emissions ..... 18
3.2.1 Modulation Spectrum ..... 18
3.2.2 In-band Spurious Emission ..... 18
3.2.3 Out-of-band Spurious Emission ..... 19
3.3 Radio Frequency Tolerance ..... 19
**4 Receiver Characteristics ..... 20**
4.1 Actual Sensitivity Level ..... 20
4.2 Interference Performance ..... 20
4.3 Out-of-Band Blocking ..... 21
4.4 Intermodulation Characteristics ..... 21
4.5 Maximum Usable Level ..... 22
4.6 Reference Signal Definition ..... 22
**5 Appendix A ..... 23**
5.1 Normal Operating Conditions (NOC) ..... 23
5.1.1 Normal Temperature and Air Humidity ..... 23
5.1.2 Nominal Supply Voltage ..... 23
5.2 Extreme Operating Conditions (EOC) ..... 23
5.2.1 Extreme Temperature And Air Humidity ..... 23
5.2.2 Extreme Supply Voltage ..... 23
**6 Appendix B - Operating Conditions ..... 24**

**Part B**
**LINK LAYER SPECIFICATION**

**1 General Description ..... 30**
1.1 Link Layer States ..... 30
1.1.1 State and Role Combination Restrictions ..... 31
1.2 Bit Ordering ..... 32
1.3 Device Address ..... 33
1.4 Physical Channel ..... 34
1.4.1 Advertising and Data Channel Indexes ..... 35

**2 Air Interface Packets ..... 36**
2.1 Packet Format ..... 36
2.1.1 Preamble ..... 36
2.1.2 Access Address ..... 36
2.1.3 PDU ..... 37
2.1.4 CRC ..... 37
2.2 Reserved for Future Use (RFU) ..... 37
2.3 Advertising Channel PDU ..... 37
2.3.1 Advertising PDUs ..... 39
2.3.1.1 ADV_IND ..... 39
2.3.1.2 ADV_DIRECT_IND ..... 40
2.3.1.3 ADV_NONCONN_IND ..... 40
2.3.1.4 ADV_SCAN_IND ..... 41
2.3.2 Scanning PDUs ..... 41
2.3.2.1 SCAN_REQ ..... 41
2.3.2.2 SCAN_RSP ..... 42
2.3.3 Initiating PDUs ..... 42
2.3.3.1 CONNECT_REQ ..... 42
2.4 Data Channel PDU ..... 44
2.4.1 LL Data PDU ..... 45
2.4.2 LL Control PDU ..... 46
2.4.2.1 LL_CONNECTION_UPDATE_REQ ..... 48
2.4.2.2 LL_CHANNEL_MAP_REQ ..... 48
2.4.2.3 LL_TERMINATE_IND ..... 49
2.4.2.4 LL_ENC_REQ ..... 49
2.4.2.5 LL_ENC_RSP ..... 49
2.4.2.6 LL_START_ENC_REQ ..... 50
2.4.2.7 LL_START_ENC_RSP ..... 50
2.4.2.8 LL_UNKNOWN_RSP ..... 50
2.4.2.9 LL_FEATURE_REQ ..... 50
2.4.2.10 LL_FEATURE_RSP ..... 50
2.4.2.11 LL_PAUSE_ENC_REQ ..... 51
2.4.2.12 LL_PAUSE_ENC_RSP ..... 51
2.4.2.13 LL_VERSION_IND ..... 51
2.4.2.14 LL_REJECT_IND ..... 51

**3 Bit Stream Processing ..... 52**
3.1 Error Checking ..... 52
3.1.1 CRC Generation ..... 52
3.2 Data Whitening ..... 53

**4 Air Interface Protocol ..... 55**
4.1 Inter Frame Space ..... 55
4.2 Timing Requirements ..... 55
4.2.1 Active Clock Accuracy ..... 55

4.2.2 Sleep Clock Accuracy ....55
4.3 Link Layer Device Filtering ....55
4.3.1 White List ....56
4.3.2 Advertising Filter Policy ....56
4.3.3 Scanner Filter Policy ....57
4.3.4 Initiator Filter Policy ....57
4.4 Non-Connected States ....57
4.4.1 Standby State ....57
4.4.2 Advertising State ....57
4.4.2.1 Advertising Channel Selection ....59
4.4.2.2 Advertising Interval ....59
4.4.2.3 Connectable Undirected Event Type ....59
4.4.2.4 Connectable Directed Event Type ....61
4.4.2.5 Scannable Undirected Event Type ....62
4.4.2.6 Non-connectable Undirected Event Type ....64
4.4.3 Scanning State ....65
4.4.3.1 Passive Scanning ....66
4.4.3.2 Active Scanning ....66
4.4.4 Initiating State ....67
4.5 Connection State ....67
4.5.1 Connection Events ....68
4.5.2 Supervision Timeout ....69
4.5.3 Connection Event Transmit Window ....70
4.5.4 Connection Setup – Master Role ....70
4.5.5 Connection Setup – Slave Role ....71
4.5.6 Closing Connection Events ....72
4.5.7 Window Widening ....73
4.5.8 Data Channel Index Selection ....74
4.5.8.1 Channel Classification ....74
4.5.8.2 Channel Selection ....74
4.5.9 Acknowledgement and Flow Control ....75
4.5.9.1 Flow Control ....77
4.6 Feature Support ....77
4.6.1 LE Encryption ....77
**5 Link Layer Control ....78**
5.1 Link Layer Control Procedures ....78
5.1.1 Connection Update Procedure ....78
5.1.2 Channel Map Update Procedure ....80
5.1.3 Encryption Procedure ....82
5.1.3.1 Encryption Start Procedure ....82
5.1.3.2 Encryption Pause Procedure ....84

5.1.4 Feature Exchange Procedure ..... 85
5.1.5 Version Exchange ..... 86
5.1.6 Termination Procedure ..... 86
5.2 Procedure Response Timeout ..... 87

**Part C**
**SAMPLE DATA**

**1 Encryption sample data ..... 91**
1.1 Encrypt Command ..... 93
1.2 Derivation of the MIC and Encrypted Data ..... 93

**Part D**
**MESSAGE SEQUENCE CHARTS**

**1 Introduction ..... 100**
1.1 Notation ..... 100
1.2 Control Flow ..... 101
1.3 Example MSC ..... 101
**2 Standby State ..... 102**
2.1 Initial Setup ..... 102
2.2 Random Device Address ..... 103
2.3 White Lists ..... 103
**3 Advertising State ..... 104**
3.1 Undirected Advertising ..... 104
3.2 Directed Advertising ..... 105
**4 Scanning State ..... 106**
4.1 Passive Scanning ..... 106
4.2 Active Scanning ..... 106
**5 Initiating State ..... 108**
5.1 Initiating a Connection ..... 108
5.2 Canceling an Initiation ..... 108
**6 Connection State ..... 110**
6.1 Sending Data ..... 110
6.2 Connection Update ..... 110
6.3 Channel Map Update ..... 111
6.4 Features Exchange ..... 111
6.5 Version Exchange ..... 112
6.6 Start Encryption ..... 113
6.7 Start Encryption without Long Term Key[#E3073] ..... 114
6.8 Start Encryption with Event Masked ..... 115
6.9 Start Encryption Without Slave Supporting Encryption ..... 115

6.10 Restart Encryption ..... 116
6.11 Disconnect ..... 116

**Part E**
**LOW ENERGY LINK LAYER SECURITY**

**1 Encryption and Authentication Overview ..... 121**

**2 CCM ..... 122**
2.1 CCM Nonce ..... 122
2.2 Counter Mode Blocks ..... 123
2.3 Encryption Blocks ..... 124

**Part F**
**DIRECT TEST MODE**

**1 Introduction ..... 128**

**2 Low Energy Test Scenarios ..... 129**
2.1 Test Sequences ..... 129
2.2 Message Sequence Charts ..... 130

**3 UART Test Interface ..... 132**
3.1 UART Interface Characteristics ..... 132
3.2 UART Functional Description ..... 132
3.3 Commands and Events ..... 133
3.3.1 Command and Event Behavior ..... 133
3.3.2 Commands ..... 133
3.4 Events ..... 134
3.4.1 LE_Test_Status_Event ..... 135
3.4.2 LE_Packet_Report_Event ..... 135
3.5 Timing – Command and Event ..... 136

# PHYSICAL LAYER SPECIFICATION

***This part of the specification describes the Bluetooth low energy physical layer.***


Ford Ex. 1052

# CONTENTS

**1 Scope ..... 14**

**2 Frequency Bands and Channel Arrangement ..... 16**

**3 Transmitter Characteristics ..... 17**

3.1 Modulation Characteristics ..... 17

3.2 Spurious Emissions ..... 18

3.2.1 Modulation Spectrum ..... 18

3.2.2 In-band Spurious Emission ..... 18

3.2.3 Out-of-band Spurious Emission ..... 19

3.3 Radio Frequency Tolerance ..... 19

**4 Receiver Characteristics ..... 20**

4.1 Actual Sensitivity Level ..... 20

4.2 Interference Performance ..... 20

4.3 Out-of-Band Blocking ..... 21

4.4 Intermodulation Characteristics ..... 21

4.5 Maximum Usable Level ..... 22

4.6 Reference Signal Definition ..... 22

**5 Appendix A ..... 23**

5.1 Normal Operating Conditions (NOC) ..... 23

5.1.1 Normal Temperature and Air Humidity ..... 23

5.1.2 Nominal Supply Voltage ..... 23

5.2 Extreme Operating Conditions (EOC) ..... 23

5.2.1 Extreme Temperature And Air Humidity ..... 23

5.2.2 Extreme Supply Voltage ..... 23

**6 Appendix B - Operating Conditions ..... 24**

# 1 SCOPE

Bluetooth Low Energy (LE) devices operate in the unlicensed 2.4 GHz ISM (Industrial Scientific Medical) band. A frequency hopping transceiver is used to combat interference and fading.

An LE radio shall have a transmitter or a receiver, or both.

The LE radio shall fulfill the stated requirements for the operating conditions declared by the equipment manufacturer (see Section 5.1 and Section 5.2).

The radio parameters shall be measured according to the methods described in the LE RF PHY Test Specification.

This specification is based on the established regulations for Europe, Japan, North America, Taiwan, South Korea and China. The standard documents listed below are only for information, and are subject to change or revision at any time.

The Bluetooth SIG maintains an online database of regulations that apply to Bluetooth technology in the 2.4 GHz ISM band, posted at https://www.bluetooth.org/regulatory/newindex.cfm.

**Europe:**

Approval Standards: European Telecommunications Standards Institute, ETSI
Documents: EN 300 328, EN 301 489, ETS 300-826

**Japan:**

Approval Standards: Japanese Radio Law, JRL
Documents: Japanese Radio Law: Article 4.3, Article 28, Article 29, Article 38

Radio Equipment Regulations: Article 5, Article 6, Article 7, Article 14, Article 24, Article 9.4, Article 49.20.1.C.2, Article 49.20.1.E.3
Radio Law Enforcement Regulations: Article 6.2, Article 6.4.4.1, Article 7

**North America:**

Approval Standards: Federal Communications Commission, FCC, USA
Documents: CFR47, Part 15: Sections 15.205, 15.209 and 15.247

Approval Standards: Industry Canada, IC, Canada
Documents: RSS-210 and RSS139

**Taiwan:**

Approval Standards: National Communications Commission, NCC
Documents: Low Power 0002 (LP0002); Low-power Radio-frequency Devices Technical Regulations

**South Korea:**

Approval Standards: Korea Communications Commission, KCC
Documents: Rules on Radio Equipment 2008-116

**China:**

Approval Standards: Ministry of Industry and Information Technology, MIIT
Documents: MIIT regulation [2002]353

# 2 FREQUENCY BANDS AND CHANNEL ARRANGEMENT

The LE system operates in the 2.4 GHz ISM band at 2400-2483.5 MHz. The LE system uses 40 RF channels. These RF channels have center frequencies 2402 + k * 2 MHz, where k = 0, ..., 39.

| Regulatory Range | RF Channels |
|---|---|
| 2.400-2.4835 GHz | f=2402+k*2 MHz, k=0, … ,39 |

*Table 2.1: Operating frequency bands*

# 3 TRANSMITTER CHARACTERISTICS

The requirements stated in this section are given as power levels at the antenna connector of the LE device. If the device does not have a connector, a reference antenna with 0 dBi gain is assumed.

Due to the difficulty in making accurate radiated power measurements, systems with an integral antenna should provide a temporary antenna connector during LE RF qualification testing.

For a transmitter, the output power level at the maximum power setting shall be within the limits defined in Table 3.1.

| Minimum Output Power | Maximum Output Power |
|---|---|
| 0.01 mW (-20 dBm) | 10 mW (+10 dBm) |

*Table 3.1: Transmission power*

The output power control of a device may be changed locally, for example to optimize the power consumption or reduce interference to other equipment.

## 3.1 MODULATION CHARACTERISTICS

The modulation is Gaussian Frequency Shift Keying (GFSK) with a bandwidth-bit period product BT=0.5. The modulation index shall be between 0.45 and 0.55. A binary one shall be represented by a positive frequency deviation, and a binary zero shall be represented by a negative frequency deviation.

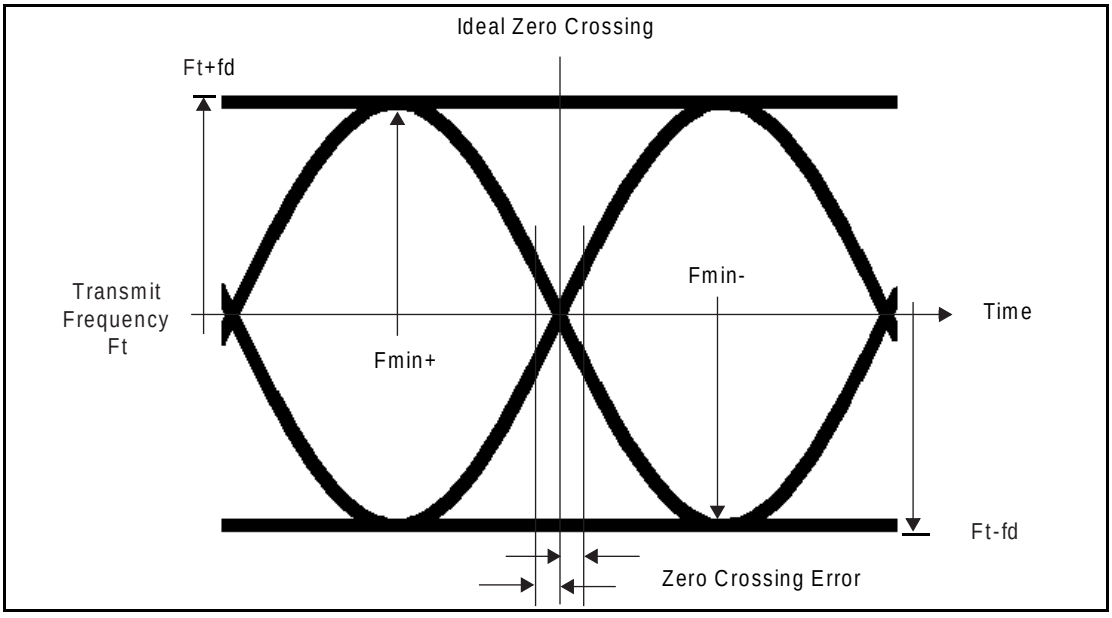



*Figure 3.1: GFSK parameters definition*

For each transmission the minimum frequency deviation,

$$F_{min} = \min \{ |F_{min+}| , F_{min-} \}$$

which corresponds to a 1010 sequence, shall be no smaller than ±80% of the frequency deviation with respect to the transmit frequency, which corresponds to a 00001111 sequence.

In addition, the minimum frequency deviation shall never be less than 185 kHz. The data transmitted has a symbol rate of 1 mega-symbols per second. The symbol timing accuracy shall be better than ±50 ppm.

The zero crossing error is the time difference between the ideal symbol period and the measured crossing time. This shall be less than ±1/8 of a symbol period.

## 3.2 SPURIOUS EMISSIONS

### 3.2.1 Modulation Spectrum

For products intended to comply with FCC part 15.247 rules, the minimum 6 dB bandwidth of the transmitter spectrum shall be at least 500 kHz using a resolution bandwidth of 100 kHz.

### 3.2.2 In-band Spurious Emission

An adjacent channel power is specified for channels at least 2 MHz from the carrier. This adjacent channel power is defined as the sum of the measured power in a 1 MHz bandwidth.

The spectrum measurement shall be performed with a 100 kHz resolution bandwidth and an average detector. The device shall transmit on an RF Channel with the center frequency M and the adjacent channel power shall be measured on a 1 MHz RF frequency N. The transmitter shall transmit a pseudo random data pattern in the payload throughout the test.

| Frequency offset | Spurious Power |
|---|---|
| 2 MHz (\|M-N\| = 2) | -20 dBm |
| 3 MHz or greater (\|M-N\| ≥ 3) | -30 dBm |

*Table 3.2: Transmit Spectrum mask*

Exceptions are allowed in up to three bands of 1 MHz width, centered on a frequency which is an integer multiple of 1 MHz. These exceptions shall have an absolute value of -20 dBm or less.

### 3.2.3 Out-of-band Spurious Emission

The equipment manufacturer is responsible for the ISM out-of-band spurious emissions requirements in the intended countries of sale.

## 3.3 RADIO FREQUENCY TOLERANCE

The deviation of the center frequency during the packet shall not exceed ±150 kHz, including both the initial frequency offset and drift. The frequency drift during any packet shall be less than 50 kHz. The drift rate shall be less than 400 Hz/µs.

The limits on the transmitter center frequency drift within a packet is shown in Table 3.3.

| Parameter | Frequency Drift |
|---|---|
| Maximum drift | ±50 kHz |
| Maximum drift rate[1] | 400 Hz/µs |

*Table 3.3: Maximum allowable frequency drifts in a packet*

1. The maximum drift rate is allowed anywhere in a packet.

# 4 RECEIVER CHARACTERISTICS

The reference sensitivity level referred to in this chapter is -70 dBm. The packet error rate corresponding to the defined bit error ratio (BER) shall be used in all receiver characteristic measurements.

## 4.1 ACTUAL SENSITIVITY LEVEL

The actual sensitivity level is defined as the receiver input level for which a BER of 0.1% is achieved.

The actual sensitivity level of the receiver shall be less than or equal to -70 dBm with any transmitter compliant to the transmitter specification specified in Section 3 together with any combination of the following allowed parameter variations:

- Initial frequency offset
- Frequency drift
- Symbol rate
- Frequency deviation

## 4.2 INTERFERENCE PERFORMANCE

The interference performance shall be measured with a wanted signal 3 dB over the reference sensitivity level. If the frequency of an interfering signal is outside of the band 2400-2483.5 MHz, the out-of-band blocking specification (see Section 4.3) shall apply. The measurement resolution shall be 1 MHz. Both the desired and the interfering signal shall be reference signals as specified in Section 4.6. The BER shall be ≤0.1% for all the signal-to-interference ratios listed in Table 4.1:

| Frequency of Interference | Ratio |
|---|---|
| Co-Channel interference, $C/I_{co\text{-}channel}$ | 21 dB |
| Adjacent (1 MHz) interference[1], $C/I_{1\ MHz}$ | 15 dB |
| Adjacent (2 MHz) interference[1], $C/I_{2\ MHz}$ | -17 dB |
| Adjacent (≥3 MHz) interference[1], $C/I_{\geq 3\ MHz}$ | -27 dB |
| Image frequency Interference[1,2,3], $C/I_{Image}$ | -9 dB |
| Adjacent (1 MHz) interference to in-band image frequency[1], $C/I_{Image\pm 1MHz}$ | -15 dB |

*Table 4.1: Interference performance*

1. If two adjacent frequency specifications from Table 4.1 are applicable to the same frequency, the more relaxed specification applies.
2. In-band image frequency
3. If the image frequency ≠ n*1 MHz, then the image reference frequency is defined as the closest n*1 MHz frequency for integer n.

Any frequencies where the requirements are not met are called spurious response RF channels. Five spurious response RF channels are allowed with a distance of ≥2 MHz from the wanted signal excluding the image frequency and the image frequency ±1MHz. On these spurious response RF channels, a relaxed interference requirement C/I = -17 dB shall be met.

## 4.3 OUT-OF-BAND BLOCKING

The out-of-band blocking applies to interfering signals outside the band 2400-2483.5 MHz. The out-of-band suppression (or rejection) shall be measured with a wanted signal 3 dB over the reference sensitivity level. The interfering signal shall be a continuous wave signal. The desired signal shall be a reference signal as specified in Section 4.6, with a center frequency of 2440 MHz. The BER shall be ≤ 0.1%. The out-of-band blocking shall fulfill the following requirements:

| Interfering Signal Frequency | Interfering Signal Power Level | Measurement resolution |
|---|---|---|
| 30 MHz – 2000 MHz | -30 dBm | 10 MHz |
| 2003 – 2399 MHz | -35 dBm | 3 MHz |
| 2484 – 2997 MHz | -35 dBm | 3 MHz |
| 3000 MHz – 12.75 GHz | -30 dBm | 25 MHz |

*Table 4.2: Out-of-band suppression (or rejection) requirements.*

Up to 10 exceptions are permitted, which are dependent upon the given RF channel and are centered at a frequency which is an integer multiple of 1 MHz:

- For at least 7 of these spurious response frequencies, a reduced interference level of at least -50 dBm is allowed in order to achieve the required BER ≤ 0.1%.
- For a maximum of 3 of the spurious response frequencies, the interference level may be lower.

## 4.4 INTERMODULATION CHARACTERISTICS

The actual sensitivity performance, BER ≤ 0.1%, shall be met under the following conditions:

- The wanted signal shall be at a frequency $f_0$ with a power level 6 dB over the reference sensitivity level. The wanted signal shall be a reference signal as specified in Section 4.6.
- A static sine wave signal shall be at a frequency $f_1$ with a power level of -50 dBm.
- An interfering signal shall be at a frequency $f_2$ with a power level of -50 dBm. The interfering signal shall be a reference signal as specified in Section 4.6.

Frequencies $f_0$, $f_1$ and $f_2$ shall be chosen such that $f_0 = 2*f_1 - f_2$ and $| f_2-f_1 | = n*1$, where n can be 3, 4, or 5. The system shall fulfill at least one of the three alternatives (n=3, 4, or 5).

## 4.5 MAXIMUM USABLE LEVEL

The maximum usable input level the receiver can operate at shall be greater than -10 dBm, and the BER shall be less than or equal to 0.1% at -10 dBm input power. The input signal shall be a reference signal as specified in Section 4.6.

## 4.6 REFERENCE SIGNAL DEFINITION

A reference signal is defined as:

Modulation = GFSK

Modulation index = 0.5 ± 1%

BT = 0.5 ± 1%

Bit Rate = 1 Mbps ±1 ppm

Modulating Data for wanted signal = PRBS9

Modulating Data for interfering signal = PRBS15

Frequency accuracy better than ±1 ppm

# 5 APPENDIX A

## 5.1 NORMAL OPERATING CONDITIONS (NOC)

### 5.1.1 Normal Temperature and Air Humidity

The normal operating temperature shall be declared by the product manufacturer. The nominal test temperature shall be within ±10°C of the normal operating temperature. The nominal operating temperature ±10°C shall not exceed the extreme limits stated in Section 5.2.1.

Operating air humidity range shall be declared by the product manufacturer. The air humidity level for the nominal test condition tests shall be within the declared NOC range.

### 5.1.2 Nominal Supply Voltage

The nominal test voltage for the equipment under normal test conditions shall be the nominal supply voltage as declared by the product manufacturer.

## 5.2 EXTREME OPERATING CONDITIONS (EOC)

### 5.2.1 Extreme Temperature And Air Humidity

The extreme temperature limits are defined as the minimum and maximum temperatures of the operating temperature range declared by the product manufacturer.

For the extreme test condition, the air humidity shall be at a level within the operating air humidity range declared by the product manufacturer (see Section 5.1.1).

### 5.2.2 Extreme Supply Voltage

The extreme supply voltages are dependent on the characteristics of the product's power supply.

If the product is designed to be operated as a part of another system or a portion of a product, the extreme voltage limits for this product or system shall be used.

The applicable upper and lower extreme supply voltages shall be declared by the product manufacturer.

# 6 APPENDIX B - OPERATING CONDITIONS

The LE radio parameters shall be compliant in the following conditions.

| Parameter | Temperature | Power supply |
|---|---|---|
| Output power | EOC | EOC |
| In-band emissions | EOC | EOC |
| Modulation characteristics | NOC | NOC |
| Carrier frequency offset and drift | EOC | EOC |
| Receiver sensitivity | EOC | EOC |
| C/I and selectivity performance | NOC | NOC |
| Blocking performance | NOC | NOC |
| Intermodulation performance | NOC | NOC |
| Maximum input signal level | NOC | NOC |

EOC = Extreme Operating Conditions

NOC = Normal Operating Conditions

Note: Validation of the LE receiver parameters is performed using the Direct Test Mode (see [Vol. 6], Part F).

# LINK LAYER SPECIFICATION

***This part of the specification describes the Bluetooth low energy Link Layer.***


Ford Ex. 1052

# CONTENTS

**1 General Description ..... 32**
1.1 Link Layer States ..... 32
1.1.1 State and Role Combination Restrictions ..... 33
1.2 Bit Ordering ..... 34
1.3 Device Address ..... 35
1.4 Physical Channel ..... 36
1.4.1 Advertising and Data Channel Indexes ..... 37

**2 Air Interface Packets ..... 38**
2.1 Packet Format ..... 38
2.1.1 Preamble ..... 38
2.1.2 Access Address ..... 38
2.1.3 PDU ..... 39
2.1.4 CRC ..... 39
2.2 Reserved for Future Use (RFU) ..... 39
2.3 Advertising Channel PDU ..... 39
2.3.1 Advertising PDUs ..... 41
2.3.1.1 ADV_IND ..... 41
2.3.1.2 ADV_DIRECT_IND ..... 42
2.3.1.3 ADV_NONCONN_IND ..... 42
2.3.1.4 ADV_SCAN_IND ..... 43
2.3.2 Scanning PDUs ..... 43
2.3.2.1 SCAN_REQ ..... 43
2.3.2.2 SCAN_RSP ..... 44
2.3.3 Initiating PDUs ..... 44
2.3.3.1 CONNECT_REQ ..... 44
2.4 Data Channel PDU ..... 46
2.4.1 LL Data PDU ..... 47
2.4.2 LL Control PDU ..... 48
2.4.2.1 LL_CONNECTION_UPDATE_REQ ..... 50
2.4.2.2 LL_CHANNEL_MAP_REQ ..... 50
2.4.2.3 LL_TERMINATE_IND ..... 51
2.4.2.4 LL_ENC_REQ ..... 51
2.4.2.5 LL_ENC_RSP ..... 51
2.4.2.6 LL_START_ENC_REQ ..... 52
2.4.2.7 LL_START_ENC_RSP ..... 52
2.4.2.8 LL_UNKNOWN_RSP ..... 52
2.4.2.9 LL_FEATURE_REQ ..... 52
2.4.2.10 LL_FEATURE_RSP ..... 52
2.4.2.11 LL_PAUSE_ENC_REQ ..... 53
2.4.2.12 LL_PAUSE_ENC_RSP ..... 53

2.4.2.13 LL_VERSION_IND ..... 53
2.4.2.14 LL_REJECT_IND ..... 53

**3 Bit Stream Processing ..... 54**
3.1 Error Checking ..... 54
3.1.1 CRC Generation ..... 54
3.2 Data Whitening ..... 55

**4 Air Interface Protocol ..... 57**
4.1 Inter Frame Space ..... 57
4.2 Timing Requirements ..... 57
4.2.1 Active Clock Accuracy ..... 57
4.2.2 Sleep Clock Accuracy ..... 57
4.3 Link Layer Device Filtering ..... 57
4.3.1 White List ..... 58
4.3.2 Advertising Filter Policy ..... 58
4.3.3 Scanner Filter Policy ..... 59
4.3.4 Initiator Filter Policy ..... 59
4.4 Non-Connected States ..... 59
4.4.1 Standby State ..... 59
4.4.2 Advertising State ..... 59
4.4.2.1 Advertising Channel Selection ..... 61
4.4.2.2 Advertising Interval ..... 61
4.4.2.3 Connectable Undirected Event Type ..... 61
4.4.2.4 Connectable Directed Event Type ..... 63
4.4.2.5 Scannable Undirected Event Type ..... 64
4.4.2.6 Non-connectable Undirected Event Type ..... 66
4.4.3 Scanning State ..... 67
4.4.3.1 Passive Scanning ..... 68
4.4.3.2 Active Scanning ..... 68
4.4.4 Initiating State ..... 69
4.5 Connection State ..... 69
4.5.1 Connection Events ..... 70
4.5.2 Supervision Timeout ..... 71
4.5.3 Connection Event Transmit Window ..... 72
4.5.4 Connection Setup – Master Role ..... 72
4.5.5 Connection Setup – Slave Role ..... 73
4.5.6 Closing Connection Events ..... 74
4.5.7 Window Widening ..... 75
4.5.8 Data Channel Index Selection ..... 76
4.5.8.1 Channel Classification ..... 76
4.5.8.2 Channel Selection ..... 76

4.5.9 Acknowledgement and Flow Control ..... 77
4.5.9.1 Flow Control ..... 79
4.6 Feature Support ..... 79
4.6.1 LE Encryption ..... 79
**5 Link Layer Control ..... 80**
5.1 Link Layer Control Procedures ..... 80
5.1.1 Connection Update Procedure ..... 80
5.1.2 Channel Map Update Procedure ..... 82
5.1.3 Encryption Procedure ..... 84
5.1.3.1 Encryption Start Procedure ..... 84
5.1.3.2 Encryption Pause Procedure ..... 86
5.1.4 Feature Exchange Procedure ..... 87
5.1.5 Version Exchange ..... 88
5.1.6 Termination Procedure ..... 88
5.2 Procedure Response Timeout ..... 89

# 1 GENERAL DESCRIPTION

## 1.1 LINK LAYER STATES

The operation of the Link Layer can be described in terms of a state machine with the following five states:

- Standby State
- Advertising State
- Scanning State
- Initiating State
- Connection State

The Link Layer state machine allows only one state to be active at a time. The Link Layer shall have at least one Link Layer state machine that supports one of Advertising State or Scanning State. The Link Layer may have multiple instances of the Link Layer state machine. Certain combinations of states and roles within multiple state machines in the Link Layer are prohibited (see Section 1.1.1).

The Link Layer in the Standby State does not transmit or receive any packets. The Standby State can be entered from any other state.

The Link Layer in the Advertising State will be transmitting advertising channel packets and possibly listening to and responding to responses triggered by these advertising channel packets. A device in the Advertising State is known as an advertiser. The Advertising State can be entered from the Standby State.

The Link Layer in the Scanning State will be listening for advertising channel packets from devices that are advertising. A device in the Scanning State is known as a scanner. The Scanning State can be entered from the Standby State.

The Link Layer in the Initiating State will be listening for advertising channel packets from a specific device(s) and responding to these packets to initiate a connection with another device. A device in the Initiating State is known as an initiator. The Initiating State can be entered from the Standby State.

The Connection State can be entered either from the Initiating State or the Advertising State. A device in the Connection State is known as being in a connection.

Within the Connection State, two roles are defined:

- Master Role
- Slave Role

When entered from the Initiating State, the Connection State shall be in the Master Role. When entered from the Advertising State, the Connection State shall be in the Slave Role.

The Link Layer in the Master Role will communicate with a device in the Slave Role and defines the timings of transmissions.

The Link Layer in the Slave Role will communicate with a single device in the Master Role.

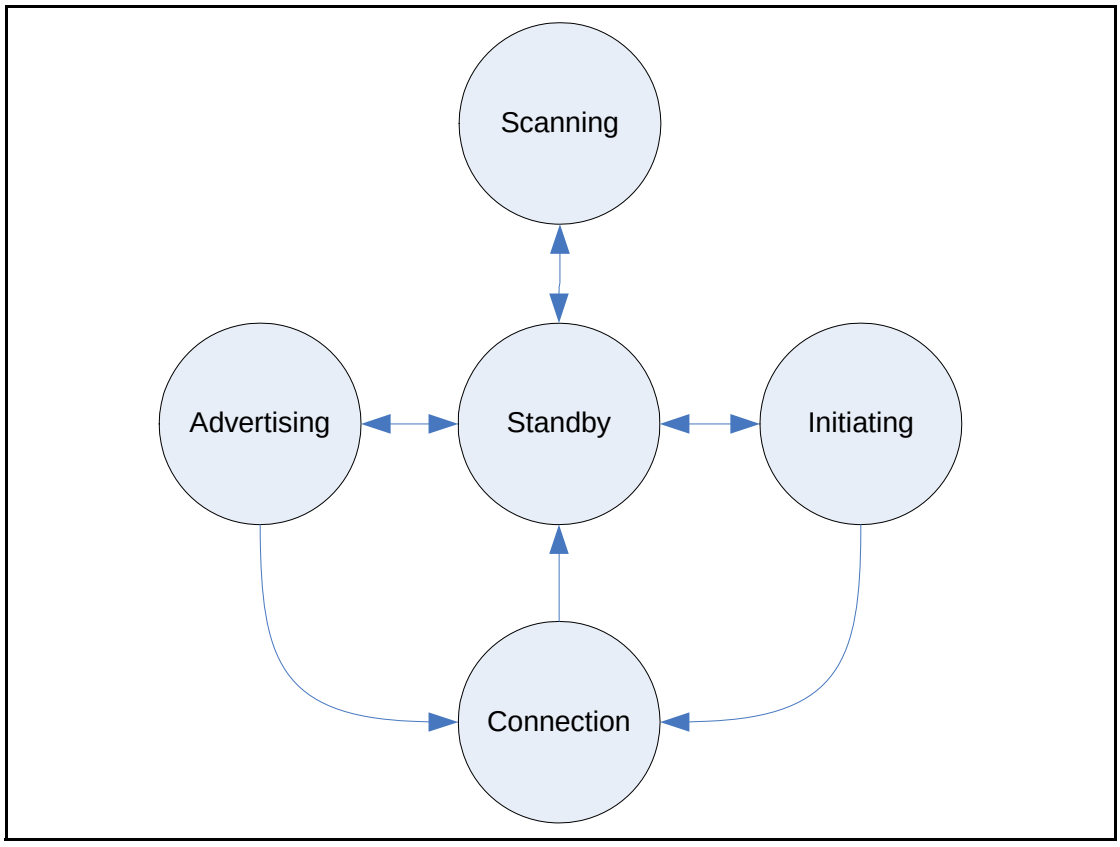



*Figure 1.1: State diagram of the Link Layer state machine*

### 1.1.1 State and Role Combination Restrictions

The Link Layer may optionally support multiple state machines. If it does support multiple state machines, the following state and role restrictions shall apply.

- The Link Layer in the Connection State shall not operate in the Master Role and Slave Role at the same time.
- The Link Layer in the Connection State operating in the Slave Role shall have only one connection.
- The Link Layer in the Connection State operating in the Master Role may have multiple connections.

- The Link Layer shall not operate in the Initiating State if the Link Layer is already operating in the Connection State in the Slave Role.
- If the Link Layer is already operating in the Connection State or Initiating State, the Link Layer shall not operate in the Advertising State with a type of advertising that could result in the Link Layer entering the Connection State in the Slave Role.

All other combinations of states may be supported. A Link Layer implementation is not required to support all the possible state combinations that are allowed by this specification.

The table below specifies the allowable and prohibited Link Layer state machine state combinations.

| **Multiple State Machine State and Roles** | | **Advertising** | **Scanning** | **Initiating** | **Connection** | |
|---|---|---|---|---|---|---|
| | | | | | **Master Role** | **Slave Role** |
| **Advertising** | | Prohibited | Allowed | Allowed* | Allowed* | Allowed* |
| **Scanning** | | Allowed | Prohibited | Allowed | Allowed | Allowed |
| **Initiating** | | Allowed* | Allowed | Prohibited | Allowed | Prohibited |
| **Connection** | **Master Role** | Allowed* | Allowed | Allowed | Allowed | Prohibited |
| | **Slave Role** | Allowed* | Allowed | Prohibited | Prohibited | Prohibited |

*Table 1.1: Allowable and prohibited Link Layer state machine state combinations*

* Only advertising packets that shall not result in the Link Layer entering Connection State in the Slave Role allowed.

## 1.2 BIT ORDERING

The bit ordering when defining fields within the packet or Protocol Data Unit (PDU) in the Link Layer specification follows the Little Endian format. The following rules apply:

- The Least Significant Bit (LSB) corresponds to $b_0$
- The LSB is the first bit sent over the air
- In illustrations, the LSB is shown on the left side

Furthermore, data fields defined in the Link Layer, such as the PDU header fields, shall be transmitted with the LSB first. For instance, a 3-bit parameter *X=3* is sent as:

$b_0b_1b_2 = 110$

Over the air, 1 is sent first, 1 is sent next, and 0 is sent last. This is shown as 110 in the specification.

Binary field values specified in this specification that follow the format 10101010b (e.g., pre-amble in Section 2.1.1 or advertising channel Access Address in Section 2.1.2) are written with the MSB to the left.

Multi-octet fields, with the exception of the Cyclic Redundancy Check (CRC) and the Message Integrity Check (MIC), shall be transmitted with the least significant octet first. Each octet within multi-octet fields, with the exception of the CRC (see Section 3.1.1), shall be transmitted in LSB first order. For example, the 48-bit addresses in the advertising channel PDUs shall be transmitted with the least significant octet first, followed by the remainder of the five octets in increasing order.

Multi-octet field values specified in this specification (e.g. the CRC initial value in Section 2.3.3.1) are written with the most significant octet to the left; for example in 0x112233445566, the octet 0x11 is the most significant octet.

## 1.3 DEVICE ADDRESS

Devices are identified using a device address. Device addresses may be either a public device address or a random device address. A public device address and a random device address are both 48 bits in length.

A device shall contain at least one type of device address and may contain both.

The public device address shall be created in accordance with section 9.2 ("48-bit universal LAN MAC addresses") of the IEEE 802-2001 standard (http://standards.ieee.org/getieee802/download/802-2001.pdf) and using a valid Organizationally Unique Identifier (OUI) obtained from the IEEE Registration Authority (see http://standards.ieee.org/regauth/oui/forms/ and sections 9 and 9.1 of the IEEE 802-2001 specification).

The public device address is divided into the following two fields:

- company_assigned field is contained in the 24 least significant bits
- company_id field is contained in the 24 most significant bits




*Figure 1.2: Format of public device address*

The random device address is divided into the following two fields:

- hash field is contained in the 24 least significant bits, as defined in [Vol. 3] Part C, Section 10.8.2.3.
- random field is contained in the 24 most significant bits, as defined in [Vol. 3] Part C, Section 10.8.2.2.

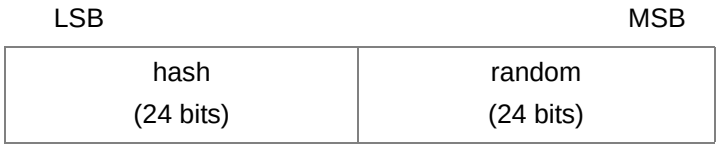



*Figure 1.3: Format of random device address*

## 1.4 PHYSICAL CHANNEL

As specified in Part A, Section 2, 40 RF Channels are defined in the 2.4GHz ISM band. These RF Channels are allocated into two LE physical channels: advertising and data. The advertising physical channel uses three RF channels for discovering devices, initiating a connection and broadcasting data. The data physical channel uses up to 37 (see Section 4.5.8) RF channels for communication between connected devices. Each of these RF Channels is allocated a unique channel index (see Section 1.4.1).

Two devices that wish to communicate use a shared physical channel. To achieve this, their transceivers must be tuned to the same RF Channel at the same time.

Given that the number of RF Channels is limited, and that many Bluetooth devices may be operating independently within the same spatial and temporal area, there is a strong likelihood of two independent Bluetooth devices having their transceivers tuned to the same RF Channel, resulting in a physical channel collision. To mitigate the unwanted effects of this collision, each transmission on a physical channel starts with an Access Address that is used as a correlation code by devices tuned to the physical channel. This Access Address is a property of the physical channel. The Access Address is present at the start of every transmitted packet.

The Link Layer uses one physical channel at a given time.

Whenever the Link Layer is synchronized to the timing, frequency and Access Address of a physical channel it is said to be 'connected' to this channel (whether or not it is actively involved in communications over the channel).

### 1.4.1 Advertising and Data Channel Indexes

Table 1.2 shows the mapping from RF Channel to Data Channel Index and Advertising Channel Index. It also shows the allocation of channel type to each RF Channel.

| RF Channel | RF Center Frequency | Channel Type | Data Channel Index | Advertising Channel Index |
|---|---|---|---|---|
| 0 | 2402 MHz | Advertising channel | | 37 |
| 1 | 2404 MHz | Data channel | 0 | |
| 2 | 2406 MHz | Data channel | 1 | |
| ... | … | Data channels | … | |
| 11 | 2424 MHz | Data channel | 10 | |
| 12 | 2426 MHz | Advertising channel | | 38 |
| 13 | 2428 MHz | Data channel | 11 | |
| 14 | 2430 MHz | Data channel | 12 | |
| ... | … | Data channels | … | |
| 38 | 2478 MHz | Data channel | 36 | |
| 39 | 2480 MHz | Advertising channel | | 39 |

*Table 1.2: Mapping of RF Channel to Data Channel Index and Advertising Channel Index*

# 2 AIR INTERFACE PACKETS

LE devices shall use the packets as defined in the following sections.

## 2.1 PACKET FORMAT

The Link Layer has only one packet format used for both advertising channel packets and data channel packets.

The packet format is shown in Figure 2.1. Each packet consists of four fields: the preamble, the Access Address, the PDU, and the CRC.

LSB MSB

| Preamble (1 octet) | Access Address (4 octets) | PDU (2 to 39 octets) | CRC (3 octets) |
|---|---|---|---|

*Figure 2.1: Link Layer packet format*

The preamble is 1 octet and the Access Address is 4 octets. The PDU range is from 2 to a maximum of 39 octets. The CRC is 3 octets.

The Preamble is transmitted first, followed by the Access Address, followed by the PDU followed by the CRC.

The shortest packet is 80 bits in length. The longest packet is 376 bits in length.

### 2.1.1 Preamble

All Link Layer packets have an eight bit preamble. The preamble is used in the receiver to perform frequency synchronization, symbol timing estimation, and Automatic Gain Control (AGC) training.

Advertising channel packets shall have 10101010b as the preamble.

The data channel packet preamble is either 10101010b or 01010101b, depending on the LSB of the Access Address. If the LSB of the Access Address is 1, the preamble shall be 01010101b, otherwise the preamble shall be 10101010b.

### 2.1.2 Access Address

The Access Address for all advertising channel packets shall be 10001110100010011011111011010110b (0x8E89BED6).

The Access Address in data channel packets shall be different for each Link Layer connection between any two devices with certain restrictions as defined below. The Access Address shall be a random 32-bit value, generated by the device in the Initiating State and sent in a connection request as defined in Section 2.3.3.1. The initiator shall ensure that the Access Address meets the following requirements:

- It shall have no more than six consecutive zeros or ones.
- It shall not be the advertising channel packets' Access Address.
- It shall not be a sequence that differs from the advertising channel packets' Access Address by only one bit.
- It shall not have all four octets equal.
- It shall have no more than 24 transitions.
- It shall have a minimum of two transitions in the most significant six bits.

### 2.1.3 PDU

The preamble and Access Address are followed by a PDU.

When a packet is transmitted in an advertising physical channel, the PDU shall be the Advertising Channel PDU as defined in Section 2.3. When a packet is transmitted in a data physical channel, the PDU shall be the Data Channel PDU as defined in Section 2.4.

### 2.1.4 CRC

At the end of every Link Layer packet there is a 24-bit CRC. It shall be calculated over the PDU. The CRC polynomial is defined in Section 3.1.1.

## 2.2 RESERVED FOR FUTURE USE (RFU)

Any field marked as RFU is reserved for future use. It shall be set to zero on transmission and ignored upon receipt.

## 2.3 ADVERTISING CHANNEL PDU

The advertising channel PDU has a 16-bit header and a variable size payload. Its format is as shown in Figure 2.2. The 16 bit Header field of the advertising channel PDU is as shown in Figure 2.3.

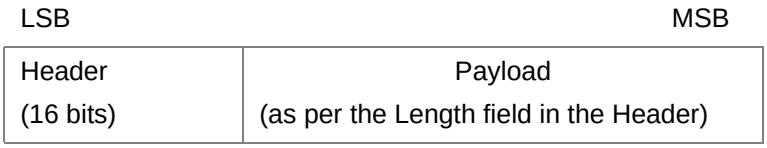



*Figure 2.2: Advertising channel PDU*

LSB MSB

| PDU Type (4 bits) | RFU (2 bits) | TxAdd (1 bit) | RxAdd (1 bit) | Length (6 bits) | RFU (2 bits) |
|---|---|---|---|---|---|

*Figure 2.3: Advertising channel PDU Header*

The PDU Type field of the advertising channel PDU that is contained in the header indicates the PDU type as defined in Table 2.1.

The TxAdd and RxAdd fields of the advertising channel PDU that are contained in the header contain information specific to the PDU type defined for each advertising channel PDU separately. If the TxAdd or RxAdd fields are not defined as used in a given PDU then they shall be considered Reserved for Future Use.

The Length field of the advertising channel PDU header indicates the payload field length in octets. The valid range of the Length field shall be 6 to 37 octets.

The Payload fields in the advertising channel PDUs are specific to the PDU Type and are defined in Section 2.3.1 through Section 2.3.3. The PDU Types marked as Reserved shall not be sent and shall be ignored upon receipt.

Within advertising channel PDUs, Advertising Data from the Host may be included in the Payload in some PDU Types. The format of this data is defined in [Vol. 3] Part C, Section 17.

| PDU Type $b_3b_2b_1b_0$ | Packet Name |
|---|---|
| 0000 | ADV_IND |
| 0001 | ADV_DIRECT_IND |
| 0010 | ADV_NONCONN_IND |
| 0011 | SCAN_REQ |
| 0100 | SCAN_RSP |
| 0101 | CONNECT_REQ |
| 0110 | ADV_SCAN_IND |
| 0111-1111 | Reserved |

*Table 2.1: Advertising channel PDU Header's PDU Type field encoding*

### 2.3.1 Advertising PDUs

The following advertising channel PDU Types are called advertising PDUs and are used in the specified events:

- ADV_IND: connectable undirected advertising event
- ADV_DIRECT_IND: connectable directed advertising event
- ADV_NONCONN_IND: non-connectable undirected advertising event
- ADV_SCAN_IND: scannable undirected advertising event

These PDUs are sent by the Link Layer in the Advertising State and received by a Link Layer in the Scanning State or Initiating State.

#### *2.3.1.1 ADV_IND*

The ADV_IND PDU has the Payload as shown in Figure 2.4. The PDU shall be used in connectable undirected advertising events. The TxAdd in the Flags field indicates whether the advertiser's address in the AdvA field is public (TxAdd = 0) or random (TxAdd = 1).

| Payload | |
|---|---|
| AdvA<br>(6 octets) | AdvData<br>(0-31 octets) |

*Figure 2.4: ADV_IND PDU Payload*

The Payload field consists of AdvA and AdvData fields. The AdvA field shall contain the advertiser's public or random device address as indicated by

TxAdd. The AdvData field may contain Advertising Data from the advertiser's Host.

### *2.3.1.2 ADV_DIRECT_IND*

The ADV_DIRECT_IND PDU has the Payload as shown in Figure 2.5. The PDU shall be used in connectable directed advertising events. The TxAdd in the Flags field indicates whether the advertiser's address in the AdvA field is public (TxAdd = 0) or random (TxAdd = 1). The RxAdd in the Flags field indicates whether the initiator's address in the InitA field is public (RxAdd = 0) or random (RxAdd = 1).

| **Payload** | |
|---|---|
| AdvA<br>(6 octets) | InitA<br>(6 octets) |

*Figure 2.5: ADV_DIRECT_IND PDU Payload*

The Payload field consists of AdvA and InitA fields. The AdvA field shall contain the advertiser's public or random device address as indicated by TxAdd. The InitA field is the address of the device to which this PDU is addressed. The InitA field shall contain the initiator's public or random device address as indicated by RxAdd.

Note: This packet does not contain any Host data.

### *2.3.1.3 ADV_NONCONN_IND*

The ADV_NONCONN_IND PDU has the Payload as shown in Figure 2.6. The PDU shall be used in non-connectable undirected advertising events. The TxAdd in the Flags field indicates whether the advertiser's address in the AdvA field is public (TxAdd = 0) or random (TxAdd = 1).

| **Payload** | |
|---|---|
| AdvA<br>(6 octets) | AdvData<br>(0-31 octets) |

*Figure 2.6: ADV_NONCONN_IND PDU Payload*

The Payload field consists of AdvA and AdvData fields. The AdvA field shall contain the advertiser's public or random device address as indicated by TxAdd. The AdvData field may contain Advertising Data from the advertiser's Host.

#### *2.3.1.4 ADV_SCAN_IND*[1]

The ADV_SCAN_IND PDU has the Payload as shown in Figure 2.7. The PDU shall be used in scannable undirected advertising events. The TxAdd in the Flags field indicates whether the advertiser's address in the AdvA field is public (TxAdd = 0) or random (TxAdd = 1).

| Payload | |
|---|---|
| AdvA<br>(6 octets) | AdvData<br>(0-31 octets) |

*Figure 2.7: ADV_SCAN_IND PDU Payload*

The Payload field consists of AdvA and AdvData fields. The AdvA field shall contain the advertiser's public or random device address as indicated by TxAdd. The AdvData field may contain Advertising Data from the advertiser's Host.

### 2.3.2 Scanning PDUs

The following advertising channel PDU Types are called scanning PDUs. They are used in the following states:

- SCAN_REQ: sent by the Link Layer in the Scanning State, received by a Link Layer in the Advertising State
- SCAN_RSP: sent by the Link Layer in the Advertising State, received by a Link Layer in the Scanning State

#### *2.3.2.1 SCAN_REQ*

The SCAN_REQ PDU has the Payload as shown in Figure 2.8. The TxAdd in the Flags field indicates whether the scanner's address in the ScanA field is public (TxAdd = 0) or random (TxAdd = 1). The RxAdd in the Flags field indicates whether the advertiser's address in the AdvA field is public (RxAdd = 0) or random (RxAdd = 1).

| Payload | |
|---|---|
| ScanA<br>(6 octets) | AdvA<br>(6 octets) |

*Figure 2.8: SCAN_REQ PDU Payload*

The Payload field consists of ScanA and AdvA fields. The ScanA field shall contain the scanner's public or random device address as indicated by TxAdd. The AdvA field is the address of the device to which this PDU is addressed.

1. ADV_DISCOVER-IND was renamed to ADV_SCAN_IND

The AdvA field shall contain the advertiser's public or random device address as indicated by RxAdd.

Note: This packet does not contain any Host Data.

#### *2.3.2.2 SCAN_RSP*

The SCAN_RSP PDU has a format as shown in Figure 2.9. The TxAdd in the Flags field indicates whether the advertiser's address in the AdvA field is public (TxAdd = 0) or random (TxAdd = 1). The Length field indicates the size of the payload (AdvA and ScanRspData) in octets.

| Payload | |
|---|---|
| AdvA<br>(6 octets) | ScanRspData<br>(0-31 octets) |

*Figure 2.9: SCAN_RSP PDU payload*

The Payload field consists of AdvA and ScanRspData fields. The AdvA field shall contain the advertiser's public or random device address as indicated by TxAdd. The ScanRspData field may contain any data from the advertiser's Host.

### 2.3.3 Initiating PDUs

The following advertising channel PDU Type is called the initiating PDU:

- CONNECT_REQ

This PDU is sent by the Link Layer in the Initiating State and received by the Link Layer in the Advertising State.

#### *2.3.3.1 CONNECT_REQ*

The CONNECT_REQ PDU has the Payload as shown in Figure 2.10. TxAdd in the Flags field indicates whether the initiator's device address in the InitA field is public (TxAdd = 0) or random (TxAdd = 1). The RxAdd in the Flags field indicates whether the advertiser's device address in the AdvA field is public (RxAdd = 0) or random (RxAdd = 1).

| Payload | | |
|---|---|---|
| InitA<br>(6 octets) | AdvA<br>(6 octets) | LLData<br>(22 octets) |

*Figure 2.10: CONNECT_REQ PDU payload*

The format of the LLData field is shown in Figure 2.11.

| LLData | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| AA (4 octets) | CRCInit (3 octets) | WinSize (1 octet) | WinOffset (2 octets) | Interval (2 octets) | Latency (2 octets) | Timeout (2 octets) | ChM (5 octets) | Hop (5 bits) | SCA (3 bits) |

*Figure 2.11: LLData field structure in CONNECT_REQ PDU's payload*

The Payload field consists of InitA, AdvA and LLData fields. The InitA field shall contain the Initiator's public or random device address as indicated by TxAdd. The AdvA field shall contain the advertiser's public or random device address as indicated by RxAdd.

The LLData consists of 10 fields:

- The AA field shall contain the Link Layer connection's Access Address determined by the Link Layer following the rules specified in Section 2.1.2.
- The CRCInit field shall contain the initialization value for the CRC calculation for the Link Layer connection, as defined in Section 3.1.1. It shall be a random value, generated by the Link Layer.
- The WinSize field shall be set to indicate the *transmitWindowSize* value, as defined in Section 4.5.3 in the following manner: *transmitWindowSize* = WinSize * 1.25 ms.
- The WinOffset field shall be set to indicate the *transmitWindowOffset*value, as defined in Section 4.5.3 in the following manner:.*transmitWindowOffset* = WinOffset * 1.25 ms.
- The Interval field shall be set to indicate the *connInterval* as defined in Section 4.5.1 in the following manner:*connInterval* = Interval * 1.25 ms.
- The Latency field shall be set to indicate the *connSlaveLatency*value, as defined in Section 4.5.1 in the following manner: *connSlaveLatency* = Latency.
- The Timeout field shall be set to indicate the *connSupervisionTimeout*value, as defined in Section 4.5.2, in the following manner: *connSupervisionTimeout* = Timeout * 10 ms.
- The ChM field shall contain the channel map indicating *Used* and *Unused* data channels. Every channel is represented with a bit positioned as per the data channel index as defined in Section 1.4.1. The LSB represents data channel index 0 and the bit in position 36 represents data channel index 36. A bit value of '0' indicates that the channel is *Unused*. A bit value of '1' indicates that the channel is *Used*. The bits in positions 37, 38 and 39 are Reserved for Future Use. Note: When mapping from RF Channels to data channel index, care should be taken to remember that there is a gap where the second advertising channel is placed.
- The Hop field shall be set to indicate the *hopIncrement* used in the data channel selection algorithm as defined in Section 4.5.8.2. It shall have a random value in the range of 5 to 16.

- The SCA field shall be set to indicate the *masterSCA* used to determine the worst case Master's sleep clock accuracy as defined in Section 4.2.2. The value of the SCA field shall be set as defined in Table 2.2.

| SCA | *masterSCA* |
|---|---|
| 0 | 251 ppm to 500 ppm |
| 1 | 151 ppm to 250 ppm |
| 2 | 101 ppm to 150 ppm |
| 3 | 76 ppm to 100 ppm |
| 4 | 51 ppm to 75 ppm |
| 5 | 31 ppm to 50 ppm |
| 6 | 21 ppm to 30 ppm |
| 7 | 0 ppm to 20 ppm |

*Table 2.2: SCA field encoding*

## 2.4 DATA CHANNEL PDU

The Data Channel PDU has a 16 bit header, a variable size payload, and may include a Message Integrity Check (MIC) field.

The Data Channel PDU is as shown in Figure 2.12.

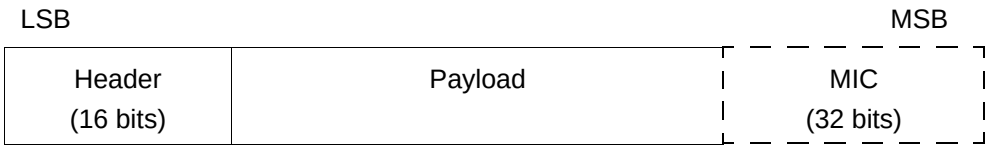



*Figure 2.12: Data Channel PDU*

The Header field of the Data Channel PDU is as shown in Figure 2.13.

| Header | | | | | | |
|---|---|---|---|---|---|---|
| LLID (2 bits) | NESN (1 bit) | SN (1 bit) | MD (1 bit) | RFU (3 bits) | Length (5 bits) | RFU (3 bits) |

*Figure 2.13: Data channel PDU header*

The 16 bit Header field consists of 5 fields that are specified in Table 2.3.

The MIC field shall not be included in an un-encrypted Link Layer connection, or in an encrypted Link Layer connection with a data channel PDU with a zero length Payload.

The MIC field shall be included in an encrypted Link Layer connection, with a data channel PDU with a non-zero length Payload and shall be calculated as specified in [Vol. 6] Part E, Section 1.

The payload format depends on the LLID field of the Header. If the LLID field is 01b or 10b, the Data Channel PDU Payload field contains an LL Data PDU as defined in Section 2.4.1. If the LLID field is 11b then the Data Channel PDU Payload field contains an LL Control PDU as defined in Section 2.4.2. An LLID field of 00b is reserved.

The NESN bit of the Header is defined in Section 4.5.9.

The SN bit of the Header is defined in Section 4.5.9.

The MD bit of the Header is defined in Section 4.5.6.

The Length field of the Header indicates the length of the Payload and MIC if included. The length field has the range of 0 to 31 octets. The Payload field shall be less than or equal to 27 octets in length. The MIC is 4 octets in length.

| Field name | Description |
|---|---|
| **LLID** | The LLID indicates whether the packet is an LL Data PDU or an LL Control PDU.<br>00b = Reserved<br>01b = LL Data PDU: Continuation fragment of an L2CAP message, or an Empty PDU.<br>10b = LL Data PDU: Start of an L2CAP message or a complete L2CAP message with no fragmentation.<br>11b = LL Control PDU |
| **NESN** | Next Expected Sequence Number |
| **SN** | Sequence Number |
| **MD** | More Data |
| **Length** | The Length field indicates the size, in octets, of the Payload and MIC, if included. |

*Table 2.3: Data channel PDU Header field*

### 2.4.1 LL Data PDU

An LL Data PDU is a data channel PDU that is used to send L2CAP data. The LLID field in the Header shall be set to either 01b or 10b.

An LL Data PDU with the LLID field in the Header set to 01b, and the Length field set to 00000b, is known as an Empty PDU. The master's Link Layer may

send an Empty PDU to the slave to allow the slave to respond with any Data Channel PDU, including an Empty PDU.

An LL Data PDU with the LLID field in the Header set to 10b shall not have the Length field set to 00000b.

### 2.4.2 LL Control PDU

An LL Control PDU is a Data Channel PDU that is used to control the Link Layer connection.

The LL Control PDU Payload is as shown in Figure 2.14.

| Payload | |
|---|---|
| Opcode<br>(1 octet) | CtrData<br>(0 – 22 octets) |

*Figure 2.14: LL control PDU payload*

An LL Control PDU shall not have the Length field set to 00000b. All LL Control PDUs have a fixed length, depending on the Opcode.

The Payload field consists of Opcode and CtrData fields.

The Opcode field identifies different types of LL Control PDU, as defined in Table 2.4.

The CtrData field in the LL Control PDU is specified by the Opcode field and is defined in Section 2.4.2.1 through Section 2.4.2.14.

| Opcode | Control PDU Name |
|---|---|
| 0x00 | LL_CONNECTION_UPDATE_REQ |
| 0x01 | LL_CHANNEL_MAP_REQ |
| 0x02 | LL_TERMINATE_IND |
| 0x03 | LL_ENC_REQ |
| 0x04 | LL_ENC_RSP |
| 0x05 | LL_START_ENC_REQ |
| 0x06 | LL_START_ENC_RSP |
| 0x07 | LL_UNKNOWN_RSP |
| 0x08 | LL_FEATURE_REQ |
| 0x09 | LL_FEATURE_RSP |
| 0x0A | LL_PAUSE_ENC_REQ |
| 0x0B | LL_PAUSE_ENC_RSP |
| 0x0C | LL_VERSION_IND |
| 0x0D | LL_REJECT_IND |
| 0x0E-0xFF | Reserved for Future Use |

*Table 2.4: LL Control PDU Opcodes*

If an LL Control PDU is received that is not used or not supported, the Link Layer shall respond with an LL_UNKNOWN_RSP PDU. The UnknownType field of the LL_UNKNOWN_RSP PDU shall be set to the value of the not used or not supported Opcode.

If an LL Control PDU is received with an invalid Opcode, i.e. the Opcode field is set to a value that is Reserved for Future Use, or with invalid CtrData fields, the Link Layer shall respond with an LL_UNKNOWN_RSP PDU. The Unknown-Type field of the LL_UNKNOWN_RSP PDU shall be set to the value of the invalid Opcode.

### 2.4.2.1 LL_CONNECTION_UPDATE_REQ

The format of the CtrData field is as shown in Figure 2.15.

| CtrData | | | | | |
|---|---|---|---|---|---|
| WinSize (1 octet) | WinOffset (2 octets) | Interval (2 octets) | Latency (2 octets) | Timeout (2 octets) | Instant (2 octets) |

*Figure 2.15: CtrData field of the LL_CONNECTION_UPDATE_REQ PDU*

The LL_CONNECTION_UPDATE_REQ CtrData consists of six fields:

- The WinSize field shall be set to indicate the *transmitWindowSize* value, as defined in Section 4.5.3 in the following manner: *transmitWindowSize* = WinSize * 1.25 ms.
- The WinOffset field shall be set to indicate the *transmitWindowOffset value,* as defined in Section 4.5.3, in the following manner: *transmitWindowOffset* = WinOffset * 1.25 ms.
- The Interval field shall be set to indicate the *connInterval* value, as defined in Section 4.4.4, in the following manner: *connInterval* = Interval * 1.25 ms.
- The Latency field shall be set to indicate the *connSlaveLatency* value, as defined by Section 4.5.1, in the following manner: *connSlaveLatency* = Latency.
- The Timeout field shall be set to indicate the *connSupervisionTimeout* value, as defined by Section 4.5.1, in the following manner*:* -connSupervisionTimeout= Timeout * 10 ms.
- The Instant field shall be set to indicate the *connInstant value, as* defined by Section 5.1.1. The Instant Field shall have a value in the range of 1 to 32767.

### 2.4.2.2 LL_CHANNEL_MAP_REQ

The format of the CtrData field is shown in Figure 2.16.

| CtrData | |
|---|---|
| ChM (5 octets) | Instant (2 octets) |

*Figure 2.16: CtrData field of the LL_CHANNEL_MAP_REQ PDU*

The LL_CHANNEL_MAP_REQ CtrData consists of two fields:

- The ChM field shall contain the channel map indicating *Used* and *Unused* data channels. Every channel is represented with a bit positioned as per the data channel index defined by Section 4.5.8. The format of this field is identical to the ChM field in the CONNECT_REQ PDU (see Section 2.3.3.1).

- The Instant field shall be set to indicate the *connInstant* value, as defined by Section 5.1.2. The Instant field shall have a value in the range of 1 to 32767.

#### *2.4.2.3 LL_TERMINATE_IND*

The format of the CtrData field is shown in Figure 2.17.

| CtrData |
|---|
| Error Code (1 octet) |

*Figure 2.17: CtrData field of the LL_TERMINATE_IND PDU*

The LL_TERMINATE_IND CtrData consists of one field:

- The Error Code field shall be set to inform the remote device why the connection is about to be terminated. See [Vol 2] Part D for details.

#### *2.4.2.4 LL_ENC_REQ*

The format of the CtrData field is shown in Figure 2.18.

| CtrData | | | |
|---|---|---|---|
| Rand (8 octets) | EDIV (2 octets) | SKDm (8 octets) | IVm (4 octets) |

*Figure 2.18: CtrData field of the LL_ENC_REQ PDU*

The LL_ENC_REQ CtrData consists of four fields:

- The Rand field contains a random number that is provided by the Host and used with EDIV (see [Vol. 3] Part H, Section 2.4.4).
- The EDIV field contains the encrypted diversifier.
- The SKDm field contains the master's portion of the session key identifier.
- The IVm field contains the master's portion of the initialization vector.

#### *2.4.2.5 LL_ENC_RSP*

The format of the CtrData field is shown in Figure 2.19.

| CtrData | |
|---|---|
| SKDs (8 octets) | IVs (4 octets) |

*Figure 2.19: CtrData field of the LL_ENC_ RSP PDU*

The LL_ENC_RSP CtrData consists of two fields.

- The SKDs field shall contain the slave's portion of the session key identifier.

- The IVs field shall contain the slave's portion of the initialization vector.

#### *2.4.2.6 LL_START_ENC_REQ*

The LL_START_ENC_REQ PDU does not have a CtrData field.

#### *2.4.2.7 LL_START_ENC_RSP*

The LL_START_ENC_RSP PDU does not have a CtrData field.

#### *2.4.2.8 LL_UNKNOWN_RSP*

The format of the CtrData field is shown in Figure 2.20.

| CtrData |
| --- |
| UnknownType<br>(1 octet) |

*Figure 2.20: CtrData field of the LL_UNKNOWN_RSP PDU*

The LL_UNKNOWN_RSP CtrData consists of one field:

- UnknownType shall contain the Opcode field value of the received LL Control PDU.

#### *2.4.2.9 LL_FEATURE_REQ*

The format of the CtrData field is shown in Figure 2.21.

| CtrData |
| --- |
| FeatureSet<br>(8 octets) |

*Figure 2.21: CtrData field of the LL_FEATURE_REQ PDU*

The LL_FEATURE_REQ CtrData consists of one field:

- FeatureSet shall contain the set of supported features of the master's Link Layer.

#### *2.4.2.10 LL_FEATURE_RSP*

The format of the CtrData field is shown in Figure 2.22.

| CtrData |
| --- |
| FeatureSet<br>(8 octets) |

*Figure 2.22: CtrData field of the LL_FEATURE_RSP PDU*

The LL_FEATURE_RSP CtrData consists of one field:

- FeatureSet shall contain the set of used features of the slave's Link Layer.

#### *2.4.2.11 LL_PAUSE_ENC_REQ*

The LL_PAUSE_ENC_REQ packet does not have a CtrData field.

#### *2.4.2.12 LL_PAUSE_ENC_RSP*

The LL_PAUSE_ENC_RSP packet does not have a CtrData field.

#### *2.4.2.13 LL_VERSION_IND*

The format of the CtrData field is shown in Figure 2.23.

| CtrData | | |
|---|---|---|
| VersNr (1 octet) | CompId (2 octets) | SubVersNr (2 octets) |

*Figure 2.23: CtrData field of the LL_VERSION_IND PDU*

The LL_VERSION_IND CtrData consists of three fields:

- VersNr field shall contain the version of the Bluetooth Controller specification (see Bluetooth Assigned Numbers).
- CompId field shall contain the company identifier of the manufacturer of the Bluetooth Controller (see Bluetooth Assigned Numbers).
- SubVersNr field shall contain a unique value for each implementation or revision of an implementation of the Bluetooth Controller.

#### *2.4.2.14 LL_REJECT_IND*

The format of the CtrData field is shown in Figure 2.24.

| CtrData |
|---|
| Error Code (1 octet) |

*Figure 2.24: CtrData field of the LL_ REJECT_IND*

Error Code shall contain the reason a request was rejected; see [Vol 2] Part D.

# 3 BIT STREAM PROCESSING

Bluetooth devices shall use the bitstream processing schemes as defined in the following sections.

Figure 3.1 shows the processes that may have to be carried out on the PDU.

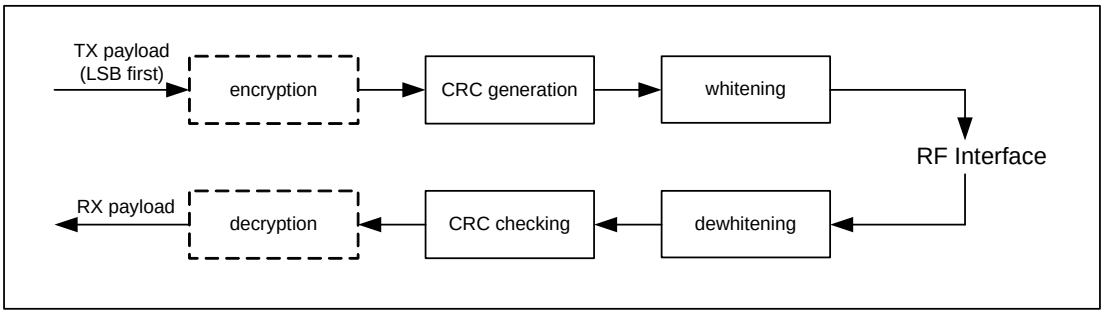



*Figure 3.1: Payload bit processes*

## 3.1 ERROR CHECKING

At packet reception, the Access Address shall be checked first. If the Access Address is incorrect, the packet shall be rejected, otherwise the packet shall be considered received. If the CRC is incorrect, the packet shall be rejected, otherwise the packet shall be considered valid. A packet shall only be processed if the packet is considered valid. A packet with an incorrect CRC may cause a connection event to continue, as specified in Section 4.5.1.

### 3.1.1 CRC Generation

The CRC shall be calculated on the PDU field in all Link Layer packets. If the PDU is encrypted, then the CRC shall be calculated after encryption of the PDU has been performed.

The CRC polynomial is a 24-bit CRC and all bits in the PDU shall be processed in transmitted order starting from the least significant bit. The polynomial has the form of $x^{24} + x^{10} + x^9 + x^6 + x^4 + x^3 + x + 1$. For every Data Channel PDU, the shift register shall be preset with the CRC initialization value set for the Link Layer connection and communicated in the CONNECT_REQ PDU. For every Advertising Channel PDU the shift register shall be preset with 0x555555.

Position 0 shall be set as the least significant bit and position 23 shall be set as the most significant bit of the initialization value. The CRC is transmitted most significant bit first, i.e. from position 23 to position 0 (see Section 1.2).

Figure 3.2 shows an example linear feedback shift register (LFSR) to generate the CRC.

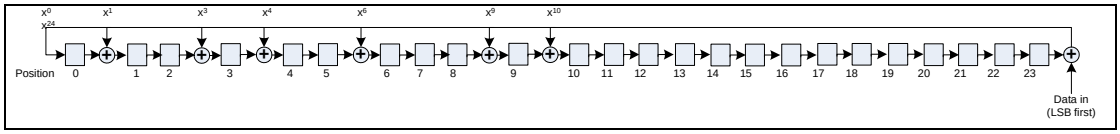



*Figure 3.2: The LFSR circuit generating the CRC*

## 3.2 DATA WHITENING

Data whitening is used to avoid long sequences of zeros or ones, e.g. 0000000b or 1111111b, in the data bit stream. Whitening shall be applied on the PDU and CRC fields of all Link Layer PDUs and is performed after the CRC in the transmitter. De-whitening is performed before the CRC in the receiver (see Figure 3.1).

The whitener and de-whitener are defined the same way, using a 7-bit linear feedback shift register with the polynomial $x^7 + x^4 + 1$. Before whitening or de-whitening, the shift register is initialized with a sequence that is derived from the channel index (data channel index or advertising channel index) in which the packet is transmitted in the following manner:

- Position 0 is set to one.
- Positions 1 to 6 are set to the channel index of the channel used when transmitting or receiving, from the most significant bit in position 1 to the least significant bit in position 6.

For example, if the channel index = 23 (0x17), the positions would be set as follows:

Position 0 = 1

Position 1 = 0

Position 2 = 1

Position 3 = 0

Position 4 = 1

Position 5 = 1

Position 6 = 1

Figure 3.3 shows an example linear feedback shift register (LFSR) to generate data whitening.

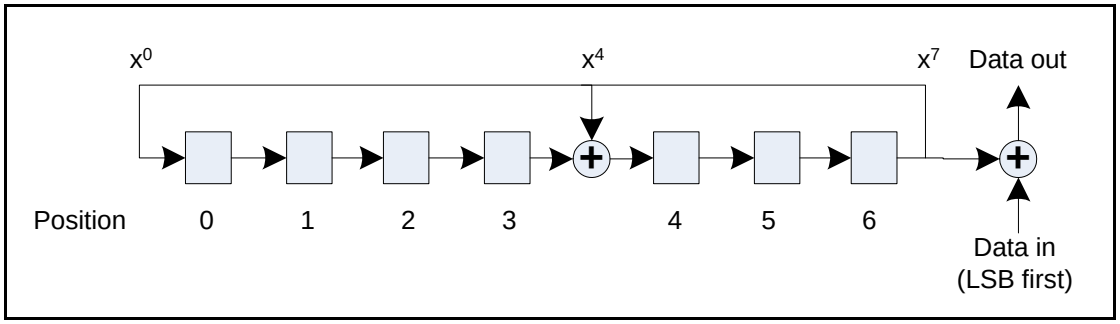



*Figure 3.3: The LFSR circuit to generate data whitening*

# 4 AIR INTERFACE PROTOCOL

The air interface protocol consists of the multiple access scheme, device discovery and link layer connection methods.

## 4.1 INTER FRAME SPACE

The time interval between two consecutive packets on the same channel index is called the Inter Frame Space. It is defined as the time from the end of the last bit of the previous packet to the start of the first bit of the subsequent packet. The Inter Frame Space is designated “T_IFS” and shall be 150 µs.

## 4.2 TIMING REQUIREMENTS

The Link Layer shall use one of two possible clock accuracies. During a connection event or advertising event the Link Layer shall use the active clock accuracy; otherwise it shall use the sleep clock accuracy.

### 4.2.1 Active Clock Accuracy

The average timing of packet transmission during a connection event is determined using the active clock accuracy, with a drift less than or equal to ±50 ppm. All instantaneous timings shall not deviate more than 2 µs from the average timing.

Note: This means that the start of a packet shall be transmitted 150±2 µs after the end of the previous packet.

### 4.2.2 Sleep Clock Accuracy

The timing of advertising events (see Section 4.4.2.2) and connection events (see Section 4.5.7) is determined using the sleep clock accuracy, with a drift less than or equal to ±500 ppm.

The instantaneous timing of the anchor point (see Section 4.5.7) shall not deviate more than 16 µs from the average timing.

Note: This means that a 1 sec connection interval with a total ±1000ppm sleep clock accuracy will give a window widening either side of the anchor point of 1ms plus 16us, assuming that the slave controller was using its sleep clock for almost the complete connection interval.

## 4.3 LINK LAYER DEVICE FILTERING

The Link Layer may perform device filtering based on the device address of the peer device. Link Layer Device Filtering is used by the Link Layer to minimize the number of devices to which it responds.

A Link Layer shall support Link Layer Device Filtering unless it only supports non-connectable advertising.

The filter policies for the Advertising State, Scanning State and Initiating State are independent of each other. When the Link Layer is in the Advertising State, the advertising filter policy shall be used. When the Link Layer is in the Scanning State, the scanning filter policy shall be used. When the Link Layer is in the Initiating State, the initiator filter policy shall be used. If the Link Layer does not support the Advertising State, Scanning State, or Initiating State, the corresponding filter policy is not required to be supported.

### 4.3.1 White List

The set of devices that the Link Layer uses for device filtering is called the White List.

A White List contains a set of White List Records used for Link Layer Device Filtering. A White List Record contains both the device address and the device address type (public or random). All Link Layers supporting Link Layer Device Filtering shall support a White List capable of storing at least one White List Record.

On reset, the White List shall be empty.

The White List is configured by the Host and is used by the Link Layer to filter advertisers, scanners or initiators. This allows the Host to configure the Link Layer to act on a request without awakening the Host.

All the device filter policies shall use the same White List.

### 4.3.2 Advertising Filter Policy

The advertising filter policy determines how the advertiser's Link Layer processes scan and/or connection requests.

When the Link Layer is using connectable directed advertising the advertising filter policy shall be ignored, otherwise the Link Layer shall use one of the following advertising filter policy modes which are configured by the Host:

- The Link Layer shall process scan and connection requests only from devices in the White List.
- The Link Layer shall process scan and connection requests from all devices (i.e. the White List is not in use). This is the default on reset.
- The Link Layer shall process scan requests from all devices and shall only process connection requests from devices that are in the White List.
- The Link Layer shall process connection requests from all devices and shall only process scan requests from devices that are in the White List.

Only one advertising filter policy mode shall be supported at a time.

### 4.3.3 Scanner Filter Policy

The scanner filter policy determines how the scanner's Link Layer processes advertising packets. The Link Layer shall use one of the following scanner filter policy modes which are configured by the Host:

- The Link Layer shall process advertising packets only from devices in the White List.
- The Link Layer shall process all advertising packets (i.e., the White List is not used). This is the default on reset.

In addition to the scanner filter policy, a connectable directed advertising packet not containing the scanner's device address shall be ignored.

Only one scanner filter policy mode shall be supported at a time.

### 4.3.4 Initiator Filter Policy

The initiator filter policy determines how an initiator's Link Layer processes advertising packets. The Link Layer shall use one of the following initiator filter policy modes which are configured by the Host:

- The Link Layer shall process connectable advertising packets from all devices in the White List.
- The Link Layer shall ignore the White List and process connectable advertising packets from a specific single device specified by the Host.

If the Link Layer receives a connectable directed advertising packet from an advertiser that is not contained in the White List or the single address specified by the Host, the connectable directed advertising packet shall be ignored.

Only one initiator filter policy mode shall be supported at a time.

## 4.4 NON-CONNECTED STATES

### 4.4.1 Standby State

The Standby State is the default state in the Link Layer. The Link Layer shall not send or receive packets in the Standby State. The Link Layer may leave the Standby State to enter the Advertising State, Scanning State or Initiator State.

### 4.4.2 Advertising State

The Link Layer shall enter the Advertising State when directed by the Host. When placed in the Advertising State, the Link Layer shall send advertising PDUs (see Section 2.3.1) in advertising events.

Each advertising event is composed of one or more advertising PDUs sent on used advertising channel indexes. The advertising event shall be closed after one advertising PDU has been sent on each of the used advertising channel indexes (see Section 4.4.2.1) or the advertiser may close an advertising event earlier to accommodate other functionality.

The time between two consecutive advertising events is defined in Section 4.4.2.2.

An advertising event can be one of the following types:

- a connectable undirected event
- a connectable directed event
- a non-connectable undirected event
- a scannable undirected event

For each advertising event type, a corresponding Advertising Channel PDU is used.

The first PDU of each advertising event shall be transmitted in the used advertising channel with the lowest advertising channel index.

The advertising event type determines the allowable response PDUs. The table below specifies the allowable responses for each advertising event.

| **Advertising Event Type** | **PDU used in this advertising event type** | **Allowable response PDUs for advertising event** | |
|---|---|---|---|
| | | SCAN_REQ | CONNECT_REQ |
| Connectable Undirected Event | ADV_IND | YES | YES |
| Connectable Directed Event | ADV_DIRECT_IND | NO | YES* |
| Non-connectable Undirected Event | ADV_NONCONN_IND | NO | NO |
| Scannable Undirected Event | ADV_SCAN_IND | YES | NO |

*Table 4.1: Advertising event types, PDUs used and allowable response PDUs*

* Only the correctly addressed initiator may respond.

If the advertiser receives a PDU for the advertising event that is not explicitly allowed it shall be ignored. If no PDU is received or the received PDU was ignored, the advertiser shall either send an advertising PDU on the next used advertising channel index or close the advertising event.

### *4.4.2.1 Advertising Channel Selection*

Advertising events use three predefined advertising channels. Advertising channel indexes are either used or unused.

The Link Layer shall use the advertising channel indexes as specified by the Host, and the used advertising channel indexes shall take effect when the Advertising State is entered.

### *4.4.2.2 Advertising Interval*

For all undirected advertising events, the time between the start of two consecutive advertising events (*T_advEvent*) is computed as follows for each advertising event:

*T_advEvent = advInterval + advDelay*

The a*dvInterval* shall be an integer multiple of 0.625 ms in the range of 20 ms to 10.24 s. If the advertising event type is either a scannable undirected event type or a non-connectable undirected event type, the *advInterval* shall not be less than 100 ms. If the advertising event type is a connectable undirected event type, the *advInterval* can be 20 ms or greater.

The *advDelay* is a pseudo-random value with a range of 0 ms to 10 ms generated by the Link Layer for each advertising event.

As illustrated in Figure 4.1, the advertising events are perturbed in time using the advDelay.

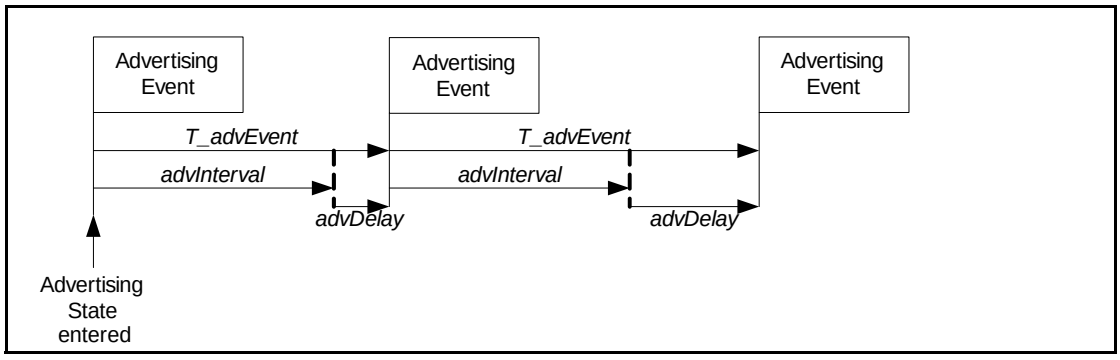



*Figure 4.1: Advertising events perturbed in time using advDelay*

### *4.4.2.3 Connectable Undirected Event Type*

When the connectable undirected advertising event type is used, advertising indications (ADV_IND PDU) are sent by the Link Layer.

The connectable undirected advertising event type allows a scanner or initiator to respond with either a scan request or connect request. A scanner may send a scan request (SCAN_REQ PDU) to request additional information about the

advertiser. An initiator may send a connect request (CONNECT_REQ PDU) to request the Link Layer to enter the Connection State.

The Link Layer shall listen on the same advertising channel index for requests from scanners or initiators.

If the advertiser receives a SCAN_REQ PDU that contains its device address from a scanner allowed by the advertising filter policy, it shall reply with a SCAN_RSP PDU on the same advertising channel index. After the SCAN_RSP PDU is sent, or if the advertising filter policy prohibited processing the SCAN_REQ PDU, the advertiser shall either move to the next used advertising channel index to send another ADV_IND PDU, or close the advertising event.

If the advertiser receives a CONNECT_REQ PDU that contains its device address, from an initiator allowed by the advertising filter policy, the Link Layer shall exit the Advertising State and transition to the Connection State in the Slave Role as defined in Section 4.5.5. If the advertising filter policy prohibited processing the received CONNECT_REQ PDU, the advertiser shall either move to the next used advertising channel index to send another ADV_IND PDU, or close the advertising event.

The time between the beginning of two consecutive ADV_IND PDUs within an advertising event shall be less than or equal to 10 ms. The advertising state shall be closed within the advertising interval.

An illustration of an advertising event using all the advertising channel indexes and in which no SCAN_REQ or CONNECT_REQ PDUs are received is shown in Figure 4.2.

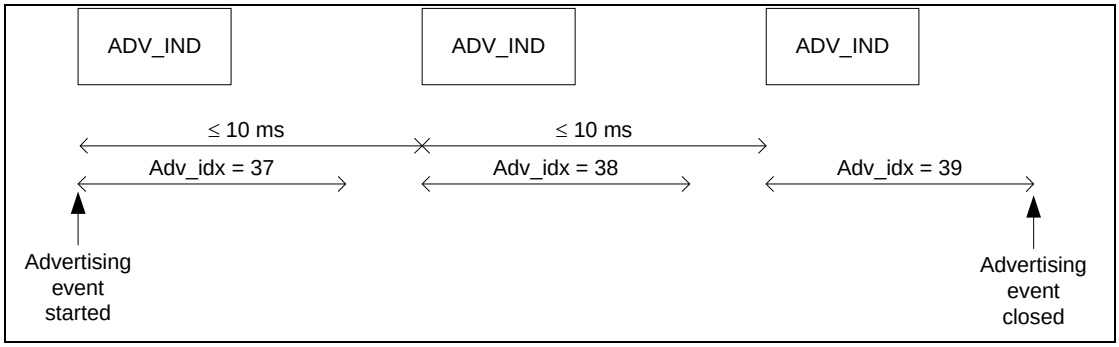



*Figure 4.2: Connectable undirected advertising event with only advertising PDUs*

Two illustrations of advertising events using all the advertising channel indexes during which a SCAN_REQ PDU is received and a SCAN_RSP PDU is sent are shown in Figure 4.3 and in Figure 4.4.

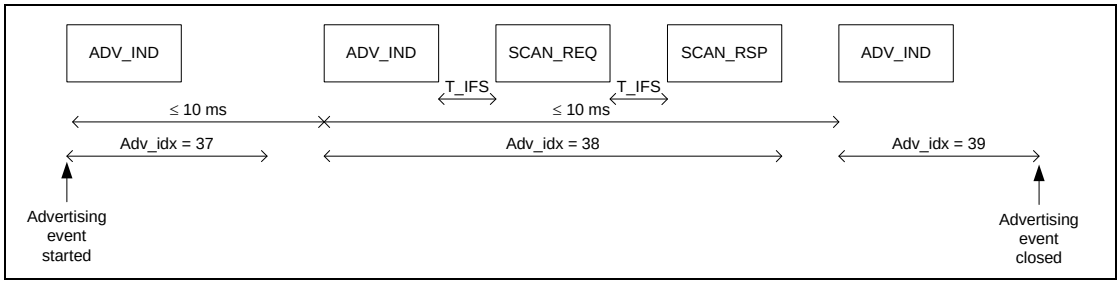



*Figure 4.3: Connectable undirected advertising event with SCAN_REQ and SCAN_RSP PDUs in the middle of an advertising event*

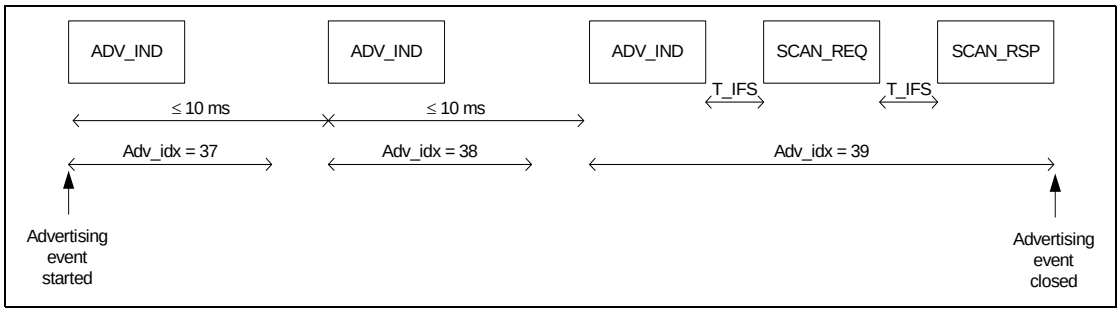
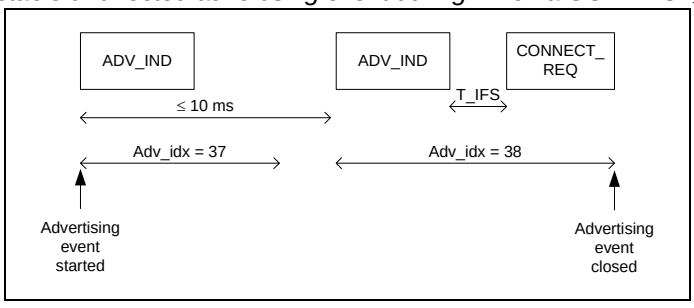



*Figure 4.4: Connectable undirected advertising event with SCAN_REQ and SCAN_RSP packets at the end of an advertising event*

Figure 4.5 illustrates an advertising event during which a CONNECT_REQ PDU is received on the second advertising channel index.

*Figure 4.5: Connectable undirected advertising event during which a CONNECT_REQ PDU is received*




### 4.4.2.4 Connectable Directed Event Type

When the connectable directed advertising event type is used, directed advertising indications (ADV_DIRECT_IND PDUs) are sent by the Link Layer.

The connectable directed advertising event type allows an initiator to respond with a connect request. An initiator may send a connect request (CONNECT_REQ PDU) to request the Link Layer to enter the Connection State.

The ADV_DIRECT_IND PDU contains both the initiator's device address and the advertiser's device address. Only the addressed initiator may initiate a Link

Layer connection with the advertiser by sending a CONNECT_REQ PDU to the advertiser.

After every ADV_DIRECT_IND PDU sent by the advertiser, the advertiser shall listen for CONNECT_REQ PDUs on the same advertising channel index. Any SCAN_REQ PDUs received shall be ignored.

If the advertiser receives a CONNECT_REQ PDU that contains its device address and the initiator device address is contained in the ADV_DIRECT_IND PDU, the Link Layer shall exit the Advertising State and transition to the Connection State in the Slave Role as defined in Section 4.5.5. Otherwise, the advertiser shall either move to the next used advertising channel index to send another ADV_DIRECT_IND PDU, or close the advertising event.

The time between the start of two consecutive ADV_DIRECT_IND PDUs sent on the same advertising channel index shall be less than or equal to 3. 75 ms.

The Link Layer shall exit the Advertising State no later than 1.28 s after the Advertising State was entered.

A sequence of five ADV_DIRECT_IND PDUs in two advertising events without CONNECT_REQ PDUs is shown in Figure 4.6 for the case in which all the advertising channels are used.

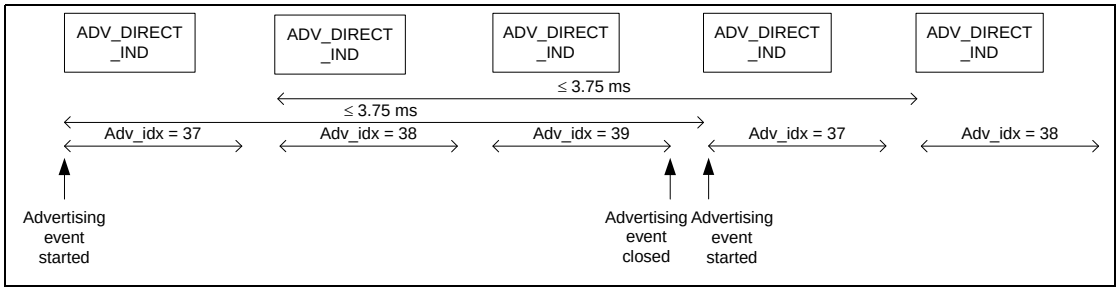



*Figure 4.6: Connectable directed advertising event type with only advertising PDUs*

Note: Connectable directed advertising is designed for cases in which fast Link Layer connection setup is essential (e.g., a reconnection). It is a power and bandwidth intensive advertising scheme that should only be used when fast connection setup is required.

### *4.4.2.5 Scannable Undirected Event Type*[1]

When the scannable undirected advertising event type is used, scannable advertising indications (ADV_SCAN_IND PDUs) packets are sent by the Link Layer.

1. Discoverable Undirected Event Type was renamed to Scannable Undirected Event Type

The scannable undirected event type allows a scanner to respond with a scan request (SCAN_REQ PDU) to request additional information about the advertiser.

The Link Layer shall listen on the same advertising channel index for requests from scanners.

If the advertiser receives a SCAN_REQ PDU that contains its device address from a scanner allowed by the advertising filter policy it shall reply with a SCAN_RSP PDU on the same advertising channel index. After the SCAN_RSP PDU is sent or if the advertising filter policy prohibited processing the SCAN_REQ PDU the advertiser shall either move to the next used advertising channel index to send another ADV_SCAN_IND PDU, or close the advertising event.

The time between the beginning of two consecutive ADV_SCAN_IND PDUs within an advertising event shall be less than or equal to 10ms. The advertising event shall be closed within the advertising interval.

The structure of an advertising event in which no SCAN_REQ PDU was received is shown in Figure 4.7 for the case in which all the advertising channels are used.

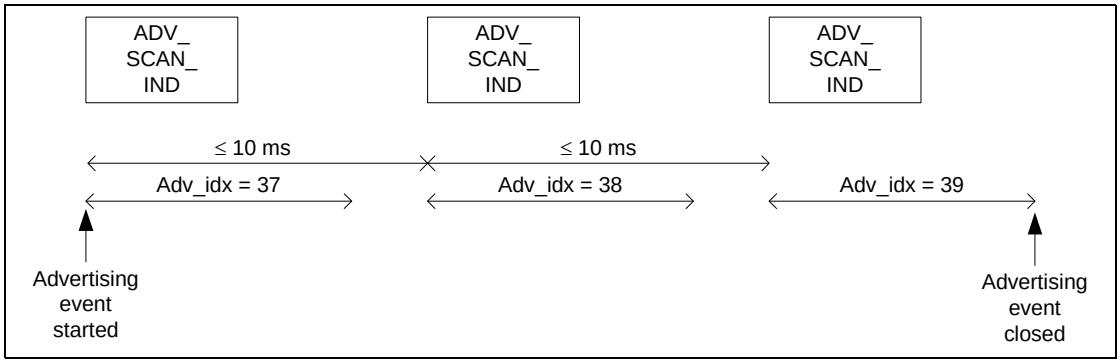



*Figure 4.7: Scannable undirected advertising event with only advertising PDUs*

Two example advertising events during which a SCAN_REQ PDU is received and a SCAN_RSP PDU is sent are shown in Figure 4.8 and in Figure 4.9 for the case in which all the advertising channels are used.

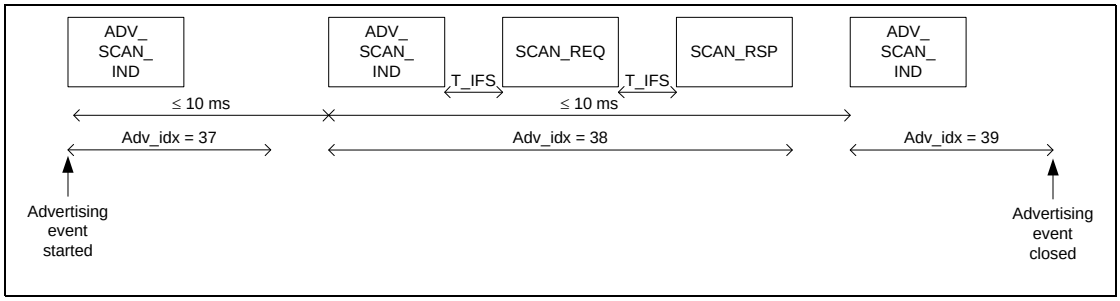



*Figure 4.8: Scannable undirected advertising event with SCAN_REQ and SCAN_RSP PDUs in the middle of an advertising event*

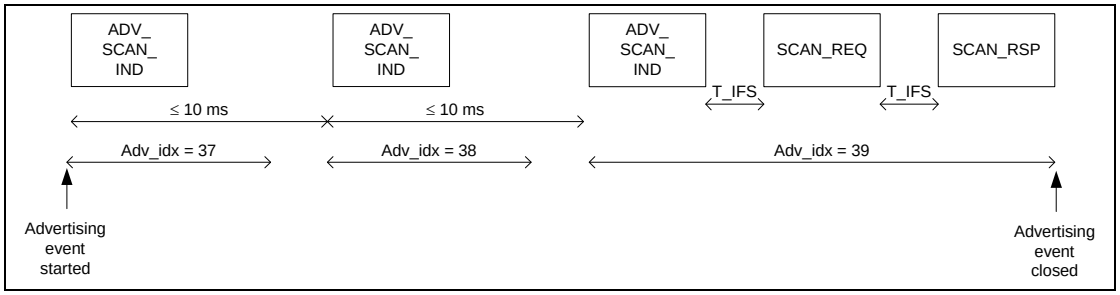



*Figure 4.9: Scannable undirected advertising event with SCAN_REQ and SCAN_RSP PDUs at the end of an advertising event*

### *4.4.2.6 Non-connectable Undirected Event Type*

When the non-connectable undirected event type is used, non-connectable advertising indications (ADV_NONCONN_IND PDU) packets are sent by the Link Layer.

The non-connectable undirected event type allows a scanner to receive information contained in the ADV_NONCONN_IND PDU from the advertiser.

The advertiser shall either move to the next used advertising channel index or close the advertising event after each ADV_NONCONN_IND PDU that is sent. The Link Layer does not listen, and therefore cannot receive any requests from scanners or initiators.

The time between the beginning of two consecutive ADV_NONCONN_IND PDUs within an advertising event shall be less than or equal to 10 ms. The advertising event shall be closed within the advertising interval.

An illustration of a non-connectable advertising event is shown in Figure 4.10 for the case in which all the advertising channels are used.

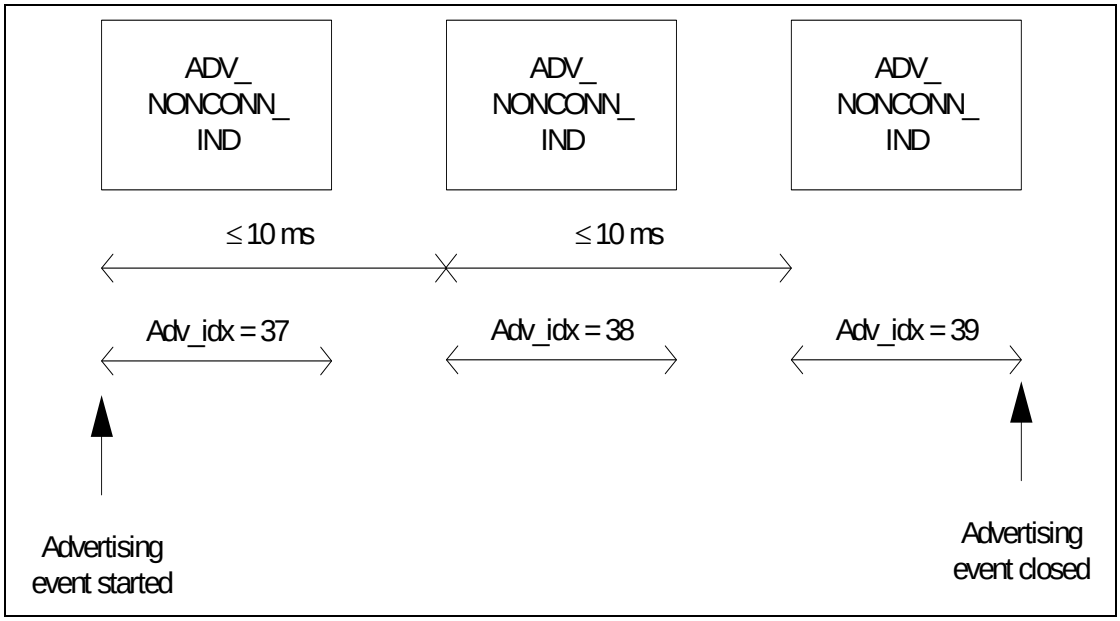



*Figure 4.10: Non-connectable undirected advertising event*

### 4.4.3 Scanning State

The Link Layer shall enter the Scanning State when directed by the Host. When scanning, the Link Layer shall listen on the advertising channel indices. There are two types of scanning, determined by the Host: passive and active.

There are no strict timing or advertising channel index selection rules for scanning.

During scanning, the Link Layer listens on an advertising channel index for the duration of the scan window, *scanWindow*. The scan interval, *scanInterval*, is defined as the interval between the start of two consecutive scan windows.

The Link Layer should listen for the complete *scanWindow* every *scanInterval* as directed by the Host unless there is a scheduling conflict. In each scan window, the Link Layer should scan on a different advertising channel index. The Link Layer shall use all the advertising channel indices.

The *scanWindow* and *scanInterval* parameters shall be less than or equal to 10.24 s. The *scanWindow* shall be less than or equal to the *scanInterval*. If the *scanWindow* and the *scanInterval* parameters are set to the same value by the Host, the Link Layer should scan continuously.

The scanner filter policy shall apply when receiving an advertising PDU when scanning.

For each non-duplicate ADV_DIRECT_IND PDU received that contains this Link Layer's device address from an advertiser the Link Layer shall send an advertising report to the Host. For each non-duplicate ADV_IND, ADV_SCAN_IND, ADV_NONCONN_IND, or SCAN_RSP PDU from an adver-

tiser, the Link Layer shall send an advertising report to the Host. The advertising report shall contain at least the advertiser's device address and advertising data or scan response data if present. Duplicate advertising reports are not required to be sent to the Host. A duplicate advertising report is an advertising report for the same device address while the Link Layer stays in the Scanning State. The advertising data may change; advertising data or scan response data is not considered significant when determining duplicate advertising reports.

#### *4.4.3.1 Passive Scanning*

When in passive scanning, the Link Layer will only receive packets; it shall not send any packets.

#### *4.4.3.2 Active Scanning*

In active scanning, the Link Layer shall listen for advertising PDUs and depending on the advertising PDU type it may request an advertiser to send additional information.

The Link Layer shall not send a SCAN_REQ PDU to an advertiser from which an ADV_DIRECT_IND PDU or ADV_NONCONN_IND PDU is received.

The Link Layer shall send at least one SCAN_REQ PDU after entering the Scanning State to advertisers from which ADV_IND or ADV_SCAN_IND PDUs are received, The Link Layer shall send further SCAN_REQ PDUs to advertisers from which ADV_IND or ADV_SCAN_IND PDUs have been received. The Link Layer should interleave SCAN_RSP PDUs to multiple advertisers.

The scanner shall run a backoff procedure to minimize collisions of SCAN_REQ PDUs from multiple scanners.

The backoff procedure uses two parameters, *backoffCount* and *upperLimit* to restrict the number of SCAN_REQ PDUs sent when collisions occur on SCAN_RSP PDUs.

Upon entering Scanning State, the *upperLimit* shall be set to one and the *backoffCount* shall be set to one.

On every received ADV_IND PDU or ADV_SCAN_IND PDU that is allowed by the scanner filter policy and a SCAN_REQ PDU is to be sent the *backoffCount* shall be decremented by one until it reaches the value of zero. The SCAN_REQ PDU shall only be sent when *backoffCount* becomes zero.

After sending a SCAN_REQ PDU the Link Layer shall listen for a SCAN_RSP PDU from that advertiser. If the SCAN_RSP PDU was not received from that advertiser, it is considered a failure otherwise it is considered a success. On every two consecutive failures, the *upperLimit* shall be doubled until it reaches

the value of 256. On every two consecutive successes, the *upperLimit* shall be halved until it reaches the value of one. After success or failure of receiving the SCAN_RSP PDU, the Link Layer shall set *backoffCount* to a new pseudo-random integer between one and *upperLimit*.

Two illustrations of advertising events using all the advertising channel indexes during which a SCAN_REQ PDU is received and a SCAN_RSP PDU is sent are shown in Figure 4.3 and in Figure 4.4.

### 4.4.4 Initiating State

The Link Layer shall enter the Initiating State when directed by the Host. When initiating, the Link Layer shall listen on the advertising channel indices.

There are no strict timing or advertising channel index selection rules for initiators.

During initiating, the Link Layer listens on an advertising channel index for the duration of the scan window, *scanWindow*. The scan interval, *scanInterval*, is defined as the interval between the start of two consecutive scan windows.

The Link Layer should listen for the complete *scanWindow* every *scanInterval* as directed by the Host unless there is a scheduling conflict. In each scan window, the Link Layer should listen on a different advertising channel index. The Link Layer shall use all the advertising channel indexes.

The *scanWindow* and *scanInterval* parameters shall be less than or equal to 10.24 s. The *scanWindow* shall be less than or equal to the *scanInterval*. If the *scanWindow* and the *scanInterval* parameters are set to the same value by the Host, the Link Layer should listen continuously.

If an ADV_IND PDU is received that is allowed by the initiator filter policy, the initiator shall send a CONNECT_REQ PDU to the advertiser. If an ADV_DIRECT_IND PDU with this Link Layer's device address is received that is allowed by the initiator filter policy, the initiator shall send a CONNECT_REQ PDU to the advertiser; otherwise it shall be ignored.

After sending the CONNECT_REQ PDU, the Link Layer shall exit the Initiating State, and shall transition to the Connection State in the Master Role as defined in Section 4.5.4.

## 4.5 CONNECTION STATE

The Link Layer enters the Connection State when an initiator sends a CONNECT_REQ PDU to an advertiser or an advertiser receives a CONNECT_REQ PDU from an initiator.

After entering the Connection State, the connection is considered to be created. The connection is not considered to be established at this point. A con-

nection is only considered to be established once a data channel packet has been received from the peer device. The only difference between a connection that is created and a connection that is established is the Link Layer connection supervision timeout value that is used (see Section 4.5.2).

When two devices are in a connection, the two devices act in different roles. A Link Layer in the Master Role is called a master. A Link Layer in the Slave Role is called a slave. The master controls the timing of a connection event. A connection event is a point of synchronization between the master and the slave.

### 4.5.1 Connection Events

The Link Layer in the Connection State shall only transmit Data Channel PDUs (see Section 2.4) in connection events. The master and slave shall determine the data channel index for each connection event as defined in Section 4.5.8. The same data channel index shall be used for all packets in the connection event. Each connection event contains at least one packet sent by the master.

During a connection event, the master and slave alternate sending and receiving packets. The connection event is considered open while both devices continue to send packets. The slave shall always send a packet if it receives a packet from the master regardless of a valid CRC match, except after multiple consecutive invalid CRC matches as specified in Section 4.5.6. The master may send a packet if it receives a packet from the slave regardless of a valid CRC match. The Length field of the Header is assumed to be correct even if the CRC match was invalid. If the master does not receive a packet from the slave, the master shall close the connection event.

The connection event can be closed by either device, as defined in Section 4.5.6.

The timing of connection events is determined by two parameters: connection event interval (*connInterval*), and slave latency (*connSlaveLatency*).

The start of a connection event is called an anchor point. At the anchor point, a master shall start to transmit a Data Channel PDU to the slave. The start of connection events are spaced regularly with an interval of *connInterval* and shall not overlap. The master shall ensure that a connection event closes at least T_IFS before the anchor point of the next connection event. The slave listens for the packet sent by its master at the anchor point.

The *connInterval* shall be a multiple of 1.25 ms in the range of 7.5 ms to 4.0 s. The *connInterval* is set by the Initiator's Link Layer in the CONNNECT_REQ PDU from the range given by the Host.

Slave latency allows a slave to use a reduced number of connection events. The connSlaveLatency parameter defines the number of consecutive connection events that the slave device is not required to listen for the master. The value of connSlaveLatency should not cause a Supervision Timeout (see Sec-

tion 4.5.2). connSlaveLatency shall be an integer in the range of 0 to ((connSupervisionTimeout / connInterval) - 1). connSlaveLatency shall also be less than 500. When connSlaveLatency is set to zero the slave device shall listen at every anchor point. If the slave does not receive a packet from the master after applying slave latency, it should listen at each anchor point and not apply slave latency until it receives a packet from the master.

Both the master and the slave shall have a 16-bit connection event counter (*connEventCounter*) for each Link Layer connection. It shall be set to zero on the first connection event sent by the master of the connection. It shall be incremented by one for each new connection event sent by the master; the *connEventCounter* shall wrap from 0xFFFF to 0x0000. This counter is used to synchronize Link Layer control procedures.

The slave shall increment *connEventCounter* for all connection events, even if it may not be listening to the master due to slave latency in those events.

### 4.5.2 Supervision Timeout

A connection can break down due to various reasons such as a device moving out of range, encountering severe interference or a power failure condition. Since this may happen without any prior warning, it is important for both the master and the slave to monitor the status of the connection.

To be able to detect link loss, both the master and the slave shall use a Link Layer connection supervision timer, $T_{LLconnSupervision}$. Upon reception of a valid packet, the timer shall be reset.

If the Link Layer connection supervision timer reaches 6 * *connInterval* before the connection is established (see Section 4.5), the connection shall be considered lost. This enables fast termination of connections that fail to establish.

Connection supervision timeout (*connSupervisionTimeout*) is a parameter that defines the maximum time between two received Data Packet PDUs before the connection is considered lost. The *connSupervisionTimeout* shall be a multiple of 10 ms in the range of 100 ms to 32.0 s and it shall be larger than (1 + *connSlaveLatency*) * *connInterval*.

If at any time in Connection State after the connection has been established and the timer reaches the *connSupervisionTimeout* value, the connection shall be considered lost.

If the connection is considered lost, the Link Layer shall not send any further packets. The Link Layer exits the Connection State and shall transition to the Standby State. The Host shall be notified of the loss of connection.

### 4.5.3 Connection Event Transmit Window

To allow the master to efficiently schedule connection events for multiple connections or other activities it may be involved in, the master has the flexibility to schedule the first connection event anchor point at a time of its choosing. The CONNECT_REQ PDU includes parameters to determine when the master can send its first packet in the Connection State to set the anchor point and when the slave must listen.

The CONNECT_REQ PDU includes three parameters used to determine the transmit window. The transmit window starts at *transmitWindowOffset* + 1.25 ms after the end of the CONNECT_REQ PDU, and the *transmitWindowSize* parameter shall define the size of the transmit window. The *connInterval* is used in the calculation of the maximum offset and size of the transmit window. *transmitWindowOffset* and *transmitWindowSize* are determined by the Link Layer.

The *transmitWindowOffset* shall be a multiple of 1.25 ms in the range of 0 ms to *connInterval*. The *transmitWindowSize* shall be a multiple of 1.25 ms in the range of 1.25 ms to the lesser of 10 ms and (*connInterval* - 1.25 ms*)*.

Therefore the start of the first packet will be no earlier than 1.25 ms + *transmitWindowOffset* and no later than 1.25 ms + *transmitWindowOffset* + *transmitWindowSize* after the end of the CONNECT_REQ PDU transmitted in the advertising channel.

### 4.5.4 Connection Setup – Master Role

After the initiator sends the CONNECT_REQ PDU the Link Layer is in Connection State in the Master Role. The master shall reset the Link Layer connection supervision timer $T_{LLconnSupervision}$. The Link Layer shall notify the Host that the connection has been created. The first connection event shall use the data channel index as specified in Section 1.4.1.

The master shall start to send the first packet within the transmit window as defined in Section 4.5.3. It is permitted that the master's first packet can extend beyond the transmit window.

The first packet sent in the Connection State by the master determines the anchor point for the first connection event, and therefore the timings of all future connection events in this connection.

The second connection event anchor point shall be *connInterval* after the first connection event anchor point. All the normal connection event transmission rules specified in Section 4.5.1 shall apply.

Two examples of the LL connection setup procedure timing from master's perspective are shown in Figure 4.11 and in Figure 4.12.

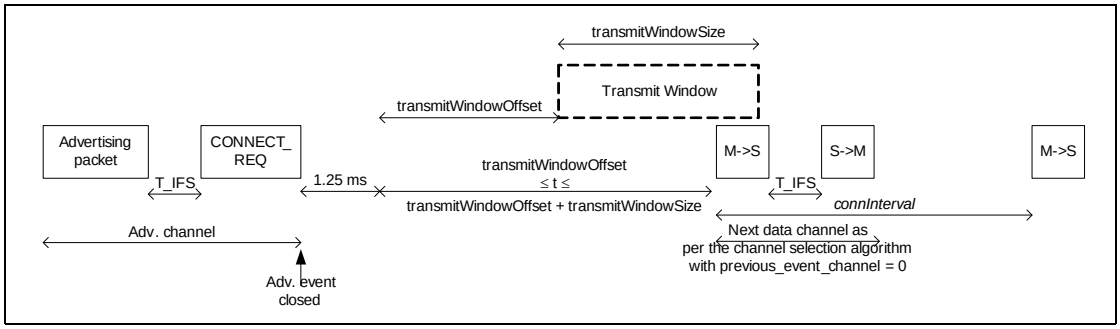



*Figure 4.11: Master's view on LL connection setup with a non-zero transmitWindowOffset*

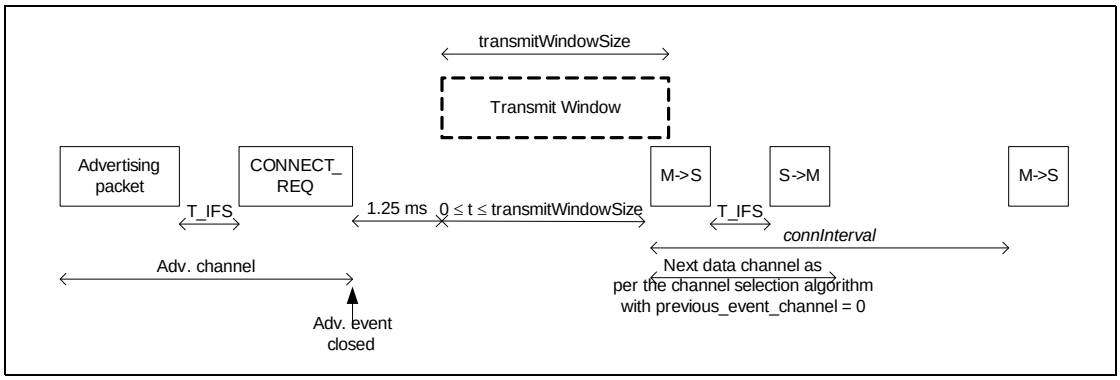



*Figure 4.12: Master's view on LL connection setup with transmitWindowOffset set to zero*

### 4.5.5 Connection Setup – Slave Role

After the advertiser receives a CONNECT_REQ PDU the Link Layer is in Connection State in the Slave Role. The slave shall reset the Link Layer connection supervision timer $T_{LLconnSupervision}$. The Link Layer shall notify the Host that the connection has been created. The first connection event shall use the data channel index as specified in Section 1.4.1.

The slave shall start to listen for the first packet within the transmit window as defined in Section 4.5.3. It is permitted that the master's first packet can extend beyond the transmit window, and therefore the slave must take this into account.

The first packet received, regardless of a valid CRC match (i.e., only the access code matches), in the Connection State by the slave determines the anchor point for the first connection event, and therefore the timings of all future connection events in this connection.

If a packet is not received in a transmit window, the slave shall attempt to receive a packet in a subsequent transmit window. A subsequent transmit window shall start *connInterval* after the start of the previous transmit window, with the same *transmitWindowSize*. The data channel index shall be the next data channel index as specified in Section 1.4.1. The *connEventCount* shall also be incremented by one.

An example of the procedure from the slave's perspective is shown in Figure 4.13 in which the slave fails to receive any part of the first packet (i.e., connEventCount = 0) from the master and acquires anchor point timing from the second packet (i.e., connEventCount = 1).

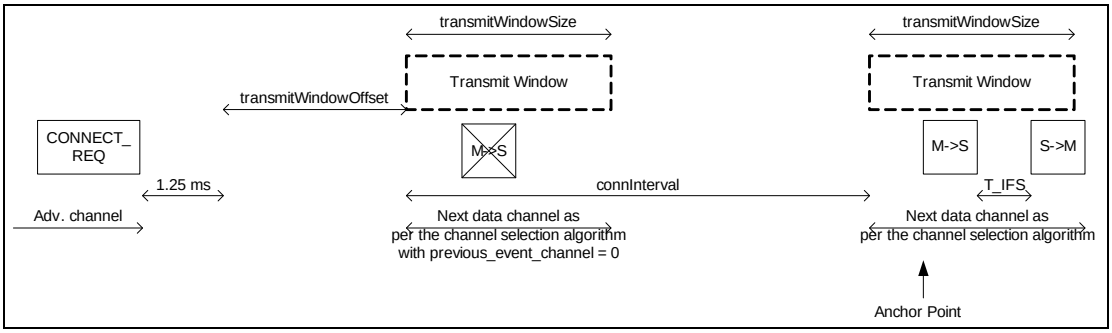



*Figure 4.13: Slave closing LL connection setup in the second LL connection event*

The slave shall be active in every connection event until the NESN is set to one in a master's packet at which point the slave can use slave subrating as defined in Section 4.5.1.

### 4.5.6 Closing Connection Events

The MD bit of the Header of the Data Channel PDU is used to indicate that the device has more data to send. If neither device has set the MD bit in their packets, the packet from the slave closes the connection event. If either or both of the devices have set the MD bit, the master may continue the connection event by sending another packet, and the slave should listen after sending its packet. If a packet is not received from the slave by the master, the master will close the connection event. If a packet is not received from the master by the slave, the slave will close the connection event.

Two consecutive packets received with an invalid CRC match within a connection event shall close the event.

MD bit usage is summarized in Table 4.2.

| | | Master | |
|---|---|---|---|
| | | **MD = 0** | **MD = 1** |
| **Slave** | **MD = 0** | Master shall not send another packet, closing the connection event.<br>Slave does not need to listen after sending its packet. | Master may continue the connection event.<br>Slave should listen after sending its packet. |
| | **MD = 1** | Master may continue the connection event.<br>Slave should listen after sending its packet. | Master may continue the connection event.<br>Slave should listen after sending its packet. |

*Table 4.2: MD bit usage for closing connection events*

### 4.5.7 Window Widening

Because of sleep clock accuracies (see Section 4.2.2), there is uncertainty in the slave of the exact timing of the master's anchor point. Therefore the slave is required to re-synchronize to the master's anchor point at each connection event where it listens for the master. If the slave receives a packet from the master regardless of a CRC match, the slave shall update its anchor point.

The slave calculates the time when the master will send the first packet of a connection event (*slaveExpectedAnchorPoint*) taking clock jittering, and in the case of connection setup or a connection parameter update the transmit window, into account. The slave shall also use the masters sleep clock accuracy (*masterSCA*) from the CONNECT_REQ PDU, together with its own sleep clock accuracy (*slaveSCA*) and the anchor point of the last connection event where it received a packet from the master (*timeSinceLastAnchor*) to calculate the time it needs to receive.

The increase in listening time is called the window widening. Assuming the clock inaccuracies are purely given in parts per million (ppm), it is calculated as follows:

*windowWidening* = ((*masterSCA* + *slaveSCA*) / 1000000) * *timeSinceLastAnchor*

During connection setup or during a connection parameter update, the slave should listen for windowWidening before the start of the transmit window and until windowWidening after the end of the transmitWindow for the master's anchor point.

At each subsequent connection event, the slave should listen for *windowWidening* before the start of the *slaveExpectedAnchorPoint* and until *windowWidening* after *slaveExpectedAnchorPoint* for the master's anchor point.

The *windowWidening* shall be smaller than ((*connInterval*/2) - T_IFS us). If the *windowWidening* reaches ((connInterval/2) - T_IFS us) in magnitude, the connection should be considered lost.

### 4.5.8 Data Channel Index Selection

#### *4.5.8.1 Channel Classification*

The master's Link Layer shall classify data channels into *used channels* (used for the connection) and *unused channels* (not used for the connection). This is called the channel map. The minimum number of used channels shall be 2.

The Host may provide channel classification information to the Link Layer. The Link Layer may use the information provided by the Host. The slave shall receive the channel map from the master in the CONNECT_REQ PDU. If the master changes the channel map it shall notify the slave as specified in Section 5.1.2.

#### *4.5.8.2 Channel Selection*

The channel selection algorithm consists of two stages: calculation of the unmapped channel index followed by mapping this index to a data channel index from the set of *used channels*.

The *unmappedChannel* and *lastUnmappedChannel* are the unmapped channel indices of two consecutive connection events. The *unmappedChannel* is the unmapped channel index for the current connection event. The *lastUnmappedChannel* is the unmapped channel index of the previous connection event. The *lastUnmappedChannel* shall be '0' for the first connection event of a connection.

At the start of a connection event, *unmappedChannel* shall be calculated using the following basic algorithm:

*unmappedChannel* = (*lastUnmappedChannel* + *hopIncrement*) mod 37

When a connection event closes, the *lastUnmappedChannel* shall be set to the value of the *unmappedChannel*.

If the *unmappedChannel* is a *used channel* according to the channel map, the channel selection algorithm shall use the *unmappedChannel* as the data channel index for the connection event.

If the *unmappedChannel* is an *unused channel* according to the channel map, the *unmappedChannel* shall be re-mapped to one of the used channels in the channel map using the following algorithm:

*remappingIndex* = *unmappedChannel* mod *numUsedChannels*

where *numUsedChannels* is the number of used channels in the channel map.

A remapping table is built that contains all the *used channels* in ascending order, indexed from zero. The *remappingIndex* is then used to select the data channel index for the connection event from the remapping table.

The complete procedure is as shown in Figure 4.14.

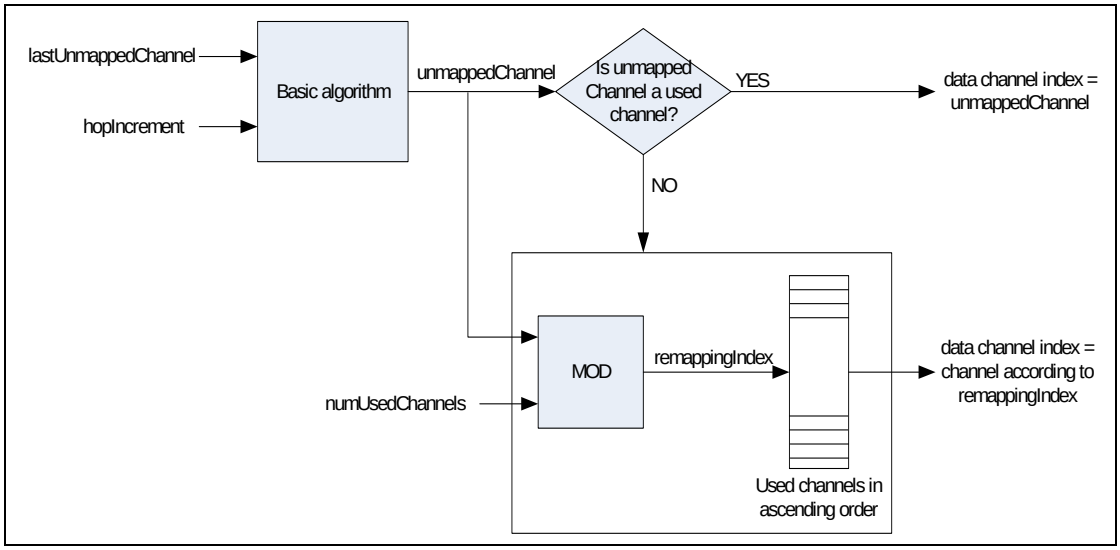



*Figure 4.14: Block diagram of data channel selection algorithm*

### 4.5.9 Acknowledgement and Flow Control

The Link Layer acknowledgement and flow control scheme shall be used in all Link Layer connections.

For each connection the Link Layer has two parameters, *transmitSeqNum* and *nextExpectedSeqNum*, each one bit in size. *transmitSeqNum* is used to identify packets sent by the Link Layer. *nextExpectedSeqNum* is used by the peer to either acknowledge the last Data Channel PDU sent, or to request resending of the last Data Channel PDU sent.

*transmitSeqNum* and *nextExpectedSeqNum* shall be set to zero upon entering the Connection State.

A new Data Channel PDU is a Data Channel PDU sent for the first time by the Link Layer. A last Data Channel PDU is a Data Channel PDU that is resent by the Link Layer. When resending a Data Channel PDU, the LLID field, the SN field and the payload of the sent Data Channel PDU shall be equal to those of the last Data Channel PDU sent by the Link Layer.

For each new Data Channel PDU that is sent, the SN bit of the Header shall be set to *transmitSeqNum*. If a Data Channel PDU is resent, then the SN bit shall not be changed.

Upon reception of a Data Channel PDU, the SN bit shall be compared to *nextExpectedSeqNum*. If the bits are different, then this is a resent Data Channel PDU, and *nextExpectedSeqNum* shall not be changed. If the bits are the same, then this is a new Data Channel PDU, and *nextExpectedSeqNum* may be incremented by one (see Section 4.5.9.1).

When a Data Channel PDU is sent, the NESN bit of the Header shall be set to *nextExpectedSeqNum*.

Upon receiving a Data Channel PDU, if the NESN bit of that Data Channel PDU is the same as *transmitSeqNum*, then the last sent Data Channel PDU has not been acknowledged and shall be resent. If the NESN bit of the Data Channel PDU is different from *transmitSeqNum*, then the last sent Data Channel PDU has been acknowledged, *transmitSeqNum* shall be incremented by one, and a new Data Channel PDU may be sent.

The above process is illustrated in Figure 4.15.

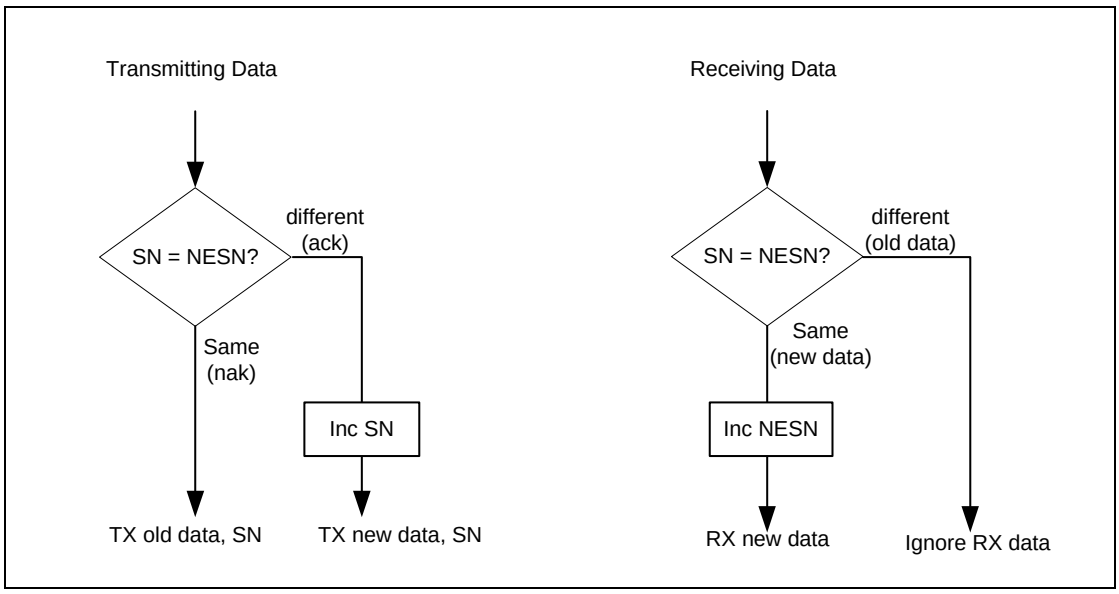



*Figure 4.15: Transmit and Receive SN and NESN flow diagram*

If a Data Channel PDU is received with an invalid CRC match, nextExpectedSeqNum shall not be changed; this means that the Data Channel PDU will not be acknowledged, causing the peer to resend the Data Channel PDU. Since the received Data Channel PDU has been rejected, the *nextExpectedSeqNum* from the peer device cannot be trusted, and therefore the last sent Data Channel PDU from this device was not acknowledged and must be retransmitted.

SN, NESN and MD bits shall be used from every received Data Channel PDU which has passed the CRC check. The Data Channel PDU payload shall be ignored on every received Data Channel PDU that has the same SN value as the previously received Data Channel PDU.

#### *4.5.9.1 Flow Control*

A Link Layer may not update nextExpectedSeqNum for reasons, including, but not limited to, lack of receive buffer space. This will cause the peer to resend the Data Channel PDU at a later time, thus enabling flow control.

## 4.6 FEATURE SUPPORT

When this information is sent from a Controller to a Host, a bit set to '0' indicates that the Link Layer Feature is not supported in this Controller; a bit set to '1' indicates that the Link Layer Feature is supported in this Controller.

When this information is sent from a Controller to a peer Controller, a bit set to '0' indicates that the Link Layer Feature shall not be used by the Controllers; a bit set to '1' indicates that the Link Layer Feature may be used by the Controllers.

The bit positions for each Link Layer Feature shall be as shown in Table 4.3. This table also shows if these bits are valid for the intended destination. If a bit is shown as not valid, using 'N', then this bit shall be ignored upon receipt.

| Bit position | Link Layer Feature | Valid from Controller to Host | Valid from Host to Controller | Valid from Controller to Controller |
|---|---|---|---|---|
| 0 | LE Encryption | Y | Y | Y |
| 1 – 63 | RFU | | | |

*Table 4.3: FeatureSet field's bit mapping to Controller features*

### 4.6.1 LE Encryption

A controller that supports LE Encryption shall support the following sections within this document:

- LL_ENC_REQ (Section 2.4.2.4)
- LL_ENC_RSP (Section 2.4.2.5)
- LL_START_ENC_REQ (Section 2.4.2.6)
- LL_START_ENC_RSP (Section 2.4.2.7)
- Encryption Start Procedure (Section 5.1.3.1)
- Encryption Pause Procedure (Section 5.1.3.2)

# 5 LINK LAYER CONTROL

The Link Layer Control Protocol (LLCP) is used to control and negotiate aspects of the operation of a connection between two Link Layers. This includes procedures for control of the connection, starting and pausing encryption and other link procedures.

Procedures have specific timeout rules as defined in Section 5.2. The Termination Procedure may be initiated at any time, even if any other Link Layer Control Procedure is currently active. For all other Link Layer Control Procedures, only one Link Layer Control Procedure shall be initiated in the Link Layer at a time per connection per device. A new Link Layer Control Procedure can be initiated only after a previous Link Layer Control Procedure has completed.

The prioritization of LL Control PDUs and LL Data PDUs is implementation specific. For example, a Host cannot assume that pending data will be sent when a termination of the link is requested without waiting for those data PDUs to be completed and indicated to the Host.

## 5.1 LINK LAYER CONTROL PROCEDURES

### 5.1.1 Connection Update Procedure

The Link Layer parameters for a connection (*connInterval*, *connSlaveLatency* and *connSupervisionTimeout*) may be updated after entering the Connection State. The master can update the connection parameters by sending an LL_CONNECTION_UPDATE_REQ PDU. The slave shall not send this PDU; the slave may request a change to the connection parameters using the L2CAP LE signaling channel.

The Link Layer of the master shall determine the *connInterval* from the interval range given by the Host ($connInterval_{min}$ and $connInterval_{max}$). The Link Layer shall indicate to the Host the selected interval value.

The Instant field of the LL_CONNECTION_UPDATE_REQ PDU shall be used to indicate the *connEventCount* when the updated parameters shall be applied; this is known as the instant. The master should allow a minimum of 6 connection events that the slave will be listening for before the instant occurs.

The connection interval used before the instant is known as $connInterval_{OLD}$. The connection interval contained in the LL_CONNECTION_UPDATE_REQ PDU and used at the instant and after, is known as $connInterval_{NEW}$.

The connection slave latency used before the instant is known as $connSlaveLatency_{OLD}$. The connection slave latency contained in the LL_CONNECTION_UPDATE_REQ PDU and used at the instant and after, is known as $connSlaveLatency_{NEW}$.

The connection supervision timeout used before the instant is known as $connSupervisionTimeout_{OLD}$. The connection supervision timeout contained in the LL_CONNECTION_UPDATE_REQ PDU and used at the instant and after, is known as $connSupervisionTimeout_{NEW}$. The connection supervision timer shall be reset at the instant.

For example, the interval between the previous connection event and the connection event at the instant will be $connInterval_{OLD}$. The interval between the connection event at the instant and the next connection event will be $connInterval_{NEW}$.

When a slave receives an LL_CONNECTION_UPDATE_REQ PDU where (Instant – connEventCount) modulo 65536 is less than 32767 and Instant is not equal to connEventCount, the slave shall listen to all the connection events until it has confirmation that the master has received its acknowledgement of the LL_CONNECTION_UPDATE_REQ PDU or connEventCount equals Instant. The slave shall also listen to the connection event where connEventCount equals Instant and the connection event before it.

When a slave receives an LL_CONNECTION_UPDATE_REQ PDU where (Instant – *connEventCount*) modulo 65536 is greater than or equal to 32767 (because the instant is in the past), the Link Layer of the slave shall consider the connection to be lost, the Link Layer shall exit the Connection State, and shall transition to the Standby State and shall notify the Host.

Note: The comparison of the *connEventCount* and the received Instant field is performed using modulo 65536 math (only values from 0 to 65535 are allowed), to handle the situation when the *connEventCount* field has wrapped.

The master may adjust the anchor point when deciding the timing of the first packet transmitted with new connection parameters. A transmit window is used, as defined in Section 4.5.3. The transmit window starts at $connInterval_{OLD}$ + *transmitWindowOffset* after the anchor point of the connection event before the instant. The *transmitWindowOffset* shall be a multiple of 1.25 ms in the range of 0 ms to $connInterval_{NEW}$. The *transmitWindowSize* shall be a multiple of 1.25 ms in the range of 1.25 ms to the lesser of 10 ms and ($connInterval_{NEW}$ - 1.25 ms).

The master shall start to send the first packet within the transmit window as defined in Section 4.5.3. It is permitted that the master's first packet can extend beyond the transmit window.

The first packet sent at the instant by the master determines the new anchor point for the connection events, and therefore the timings of all future connection events in this connection.

The next connection event anchor point shall be $connInterval_{NEW}$ after the connection event anchor point at the Instant. All the normal connection event transmission rules specified in Section 1.4.1, shall apply.

An example of the connection update procedures is shown in Figure 5.1.

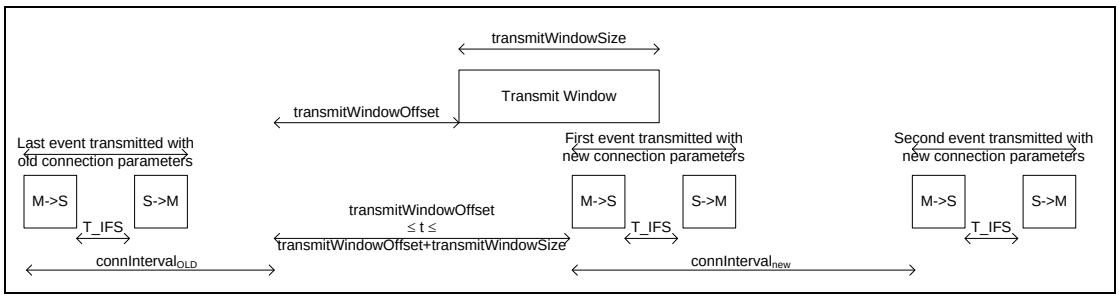



*Figure 5.1: Connection event timing in the case of connection parameter update*

At the start of the transmit window, the Link Layer shall reset $T_{LLconnSupervision}$.

If the Link Layer of the master transmits an LL_CONNECTION_UPDATE_REQ PDU autonomously, for example without being requested to by the Host, the Latency and Timeout parameters shall not be changed and shall remain the same as in the last LL_CONNECTION_UPDATE_REQ or CONNECT_REQ PDU, any of the other parameters (*transmitWindowSize*, *transmitWindowOffset*, *connInterval*, Instant) may be changed within the restrictions given above. Note: Autonomous updates can be used to change the anchor points to allow the master to change the scheduling of the connection due to other activities.

The Link Layer shall notify its Host if any of the three connection parameters have changed. If no connection parameters are changed, the Host would not be notified; this is called an anchor point move.

The procedure is complete when the instant has passed, and the new connection event parameters have been applied.

### 5.1.2 Channel Map Update Procedure

The Link Layer parameter for channel map (*channelMap*) may be updated after entering the Connection State. The master can update the channel map by sending an LL_CHANNEL_MAP_REQ PDU. The slave shall not send this PDU.

The Instant field of the LL_CHANNEL_MAP_REQ PDU shall be used to indicate the *connEventCount* when the $channelMap_{NEW}$ shall be applied; this is known as the instant. The master should allow a minimum of 6 connection events that the slave will be listening for before the instant occurs.

The channel map used before the instant is known as $channelMap_{OLD}$. The channel map contained in the LL_CHANNEL_MAP_REQ PDU and used at the instant and after, is known as $channelMap_{NEW}$.

When a slave receives an LL_CHANNEL_MAP_REQ PDU where (Instant – *connEventCount*) modulo 65536 is less than 32767, the slave shall listen to all the connection events until it has confirmation that the master has received its acknowledgement of the LL_CHANNEL_MAP_REQ PDU or *connEventCount* equals Instant.

When a slave receives an LL_CHANNEL_MAP_REQ where (Instant – *connEventCount*) modulo 65536 is greater than or equal to 32767 (because the instant is in the past), the Link Layer of the slave shall consider the connection to be lost and shall notify the Host.

Note: The comparison of the *connEventCount* and the received Instant field is performed using modulo 65536 math (only values from 0 to 65535 are allowed), to cope with the cases when the *connEventCount* field has wrapped.

When *connEventCount* is equal to the Instant field, the $channelMap_{NEW}$ shall be the current *channelMap*. The *lastUnmappedChannel* shall not be reset. If the *unmappedChannel* is an unused channel, then the $channelMap_{NEW}$ will be used when remapping. The only parameter that changes is the *channelMap*.

For example:

At connection set-up:

- initial $channelMap_{OLD}$: 0x1FFFFFFFFF (i.e., all channels enabled)
- initial *hopIncrement*: 10 (decimal)

An LL_CHANNEL_MAP_REQ PDU with the following parameters is then issued:

- Instant: 100 (decimal). Assume that no connection event count wrap-around occurred since the start of the connection.
- $channelMap_{NEW}$: 0x1FFFFFF7FF (i.e. all channels enabled except channel 11)

Channels used:

- *connEventCount* 99 --> data channel index 1 ($channelMap_{OLD}$)
- *connEventCount* 100 --> data channel index 12 (remapped from 11) ($channelMap_{NEW}$)
- *connEventCount* 101 --> data channel index 21 ($channelMap_{NEW}$)

The procedure is complete when the instant has passed, and the new channel map has been applied.

### 5.1.3 Encryption Procedure

The Link Layer, upon request from the Host, can enable the encryption of packets after entering the Connection State.

If the connection is not encrypted, the Link Layer shall only use the encryption start procedure.

If the connection is encrypted, the Link Layer shall first use the encryption pause procedure followed by the encryption start procedure.

#### *5.1.3.1 Encryption Start Procedure*

To enable encryption, two parameters must be exchanged, IV and SKD. Both are composed of two parts, a master part and a slave part, and exchanged in LL_ENC_REQ and LL_ENC_RSP PDUs. After these are exchanged, and the Host has notified the Link Layer of the Long Term Key to be used on this connection, encryption can be started using a three way handshake, using LL_START_ENC_REQ and LL_START_ENC_RSP PDUs.

To start encryption, the Link Layer of the master shall generate the master's part of the initialization vector (IVm) and the master's part of the session key diversifier (SKDm).

The Link Layer of the master shall finalize the sending of the current Data Channel PDU and may finalize the sending of additional Data Channel PDUs queued in the Controller. After these Data Channel PDUs are acknowledged, the Link Layer of the master shall only send Empty PDUs or LL_ENC_REQ, LL_START_ENC_REQ, LL_START_ENC_RSP, LL_TERMINATE_IND or LL_REJECT_IND PDUs.

The Link Layer of the master shall then send an LL_ENC_REQ PDU; the Rand and EDIV fields are provided by the Host.

If encryption is not supported by the Link Layer of the slave, the Link Layer of the slave shall send an LL_REJECT_IND PDU with the error code set to "Unsupported Remote Feature / Unsupported LMP Feature" (0x1A). The Link Layer of the master receiving the LL_REJECT_IND PDU shall notify the Host. The Link Layer of the master can now send LL Data Packets and LL Control Packets; these packets will not be encrypted. This procedure is complete in the master when the master receives the LL_REJECT_IND PDU from the slave. The procedure is complete in the slave when the acknowledgement for the LL_REJECT_IND PDU is received from the master.

Otherwise, when the Link Layer of the slave receives an LL_ENC_REQ PDU it shall generate the slave's part of the initialization vector (IVs) and the slave's part of the session key diversifier (SKDs), and notify the Host with the Rand and EDIV fields.

The Link Layer of the slave shall finalize the sending of the current Data Channel PDU and may finalize the sending of additional Data Channel PDUs queued in the Controller. After these Data Channel PDUs are acknowledged, the Link Layer of the slave is only allowed to send Empty PDUs or LL_ENC_RSP, LL_START_ENC_REQ, LL_START_ENC_RSP, LL_TERMINATE_IND or LL_REJECT_IND PDUs.

The Link Layer of the slave shall then send an LL_ENC_RSP PDU.

Each Link Layer shall combine the initialization vector parts and session key diversifier parts in the following manner:

SKD = SKDm || SKDs

IV = IVm || IVs

The SKDm is concatenated with the SKDs. The least significant octet of SKDm becomes the least significant octet of SKD. The most significant octet of SKDs becomes the most significant octet of SKD.

The IVm is concatenated with the IVs. The least significant octet of IVm becomes the least significant octet of IV. The most significant octet of IVs becomes the most significant octet of IV.

The Long Term Key is provided by the Host to the Link Layer in the master and slave, and one of the following three actions shall occur:

- If this procedure is being performed after a Pause Encryption Procedure, and the Host does not provide a Long Term Key, the slave shall perform the Termination Procedure with the error code “PIN or key Missing.”
- If the Host does not provide a Long Term Key, either because the event to the Host was masked out or if the Host indicates that a key is not available, the slave shall send an LL_REJECT_IND PDU with the error code set to “PIN or key Missing.” Upon receiving an LL_REJECT_IND PDU, the Link Layer shall notify the Host. The Link Layer can now send LL Data PDUs and LL Control PDUs; these packets will not be encrypted. This procedure is complete in the master when the master receives the LL_REJECT_IND PDU from the slave. The procedure is completed in the slave when the acknowledgement has been received for the LL_REJECT_IND PDU from the master.
- If the Host does provide a Long Term Key, the Link Layer of the slave shall respond to the LL_ENC_REQ PDU from the master with an LL_ENC_RSP PDU. The Link Layer shall also calculate *sessionKey* using the encryption engine with LTK as the key, and SKD as the plain text input. *sessionKey* shall be set to the output of the encryption engine.

The *sessionKey* shall be used as the key for the encryption engine for all encrypted packets.

After *sessionKey* has been calculated, the Link Layer of the slave shall send an LL_START_ENC_REQ PDU. This packet shall be sent unencrypted, and the Link Layer shall be set up to receive an encrypted packet in response.

When the Link Layer of the master receives an LL_START_ENC_REQ PDU it shall send an LL_START_ENC_RSP PDU. This PDU shall be sent encrypted and set up to receive encrypted.

When the Link Layer of the slave receives an LL_START_ENC_RSP PDU it shall transmit an LL_START_ENC_RSP PDU. This packet shall be sent encrypted.

When the Link Layer of the master receives the LL_START_ENC_RSP PDU, the connection is encrypted. The Link Layer can now send LL Data PDUs and LL Control PDUs; these PDUs will be encrypted.

The Link Layers shall notify the Hosts that the connection is encrypted.

The procedure is complete in the master when the master receives the LL_START_ENC_RSP PDU from the slave. The procedure is complete in the slave when the slave receives the LL_START_ENC_RSP PDU from the master.

#### *5.1.3.2 Encryption Pause Procedure*

To enable a new encryption key to be used without disconnecting the link, encryption must be disabled and then enabled again. During the pause, data PDUs shall not be sent unencrypted to protect the data.

The Link Layer of the master shall finalize the sending of the current Data Channel PDU and may finalize the sending of additional Data Channel PDUs queued in the Controller. After these Data Channel PDUs are acknowledged, the Link Layer of the master shall only send Empty PDUs or LL_PAUSE_ENC_REQ or LL_TERMINATE_IND PDUs.

The Link Layer of the master shall then send an LL_PAUSE_ENC_REQ PDU.

When the Link Layer of the slave receives an LL_PAUSE_ENC_REQ PDU it shall finalize the sending of the current Data Channel PDU and may finalize the sending of additional Data Channel PDUs queued in the Controller. After these Data Channel PDUs are acknowledged, the Link Layer of the slave is only allowed to send Empty PDUs or LL_PAUSE_ENC_RSP or LL_TERMINATE_IND PDUs.

The Link Layer of the slave shall then send an LL_PAUSE_ENC_RSP PDU. This packet shall be sent encrypted, and Link Layer shall be set up to receive unencrypted.

When the Link Layer of the master receives an LL_PAUSE_ENC_RSP PDU it shall set up to send and receive unencrypted. It shall then send an LL_PAUSE_ENC_RSP PDU to the slave unencrypted.

When the Link Layer of the slave receives an LL_PAUSE_ENC_RSP PDU it shall set up to also send unencrypted.

The encryption start procedure shall now be used to re-enable encryption using a new session key.

### 5.1.4 Feature Exchange Procedure

The Link Layer parameter for the current supported feature set (featureSet) may be exchanged after entering the Connection State. The master can initiate this procedure with an LL_FEATURE_REQ PDU, and the slave responds with an LL_FEATURE_RSP PDU. The slave cannot initiate this procedure.

The featureSet information may be cached. A Link Layer should not request this information on every connection if the information has been cached for this device. Cached information for a device may not be authoritative, and therefore an implementation must be able to accept the LL_UNKNOWN_RSP PDU if use of a feature is attempted that is not currently supported or used by the peer.

$featureSet_M$ is the feature capabilities of the Link Layer of the master. When the Link Layer of the master sends an LL_FEATURE_REQ PDU the Feature-Set field shall be set to $featureSet_M$.

$featureSet_S$ is the feature capabilities of the Link Layer of the Slave.

The $featureSet_{USED}$ is the logical AND of $featureSet_M$ and $featureSet_S$. When the Link Layer of the slave sends an LL_FEATURE_RSP PDU the FeatureSet field shall be set to $featureSet_{USED}$.

The Link Layer of the master sends an LL_FEATURE_REQ PDU. This can be sent on request from the Host or autonomously.

When the Link Layer of the slave receives an LL_FEATURE_REQ PDU it shall send an LL_FEATURE_RSP PDU. The Link Layer of the slave shall only use procedures that are indicated in $featureSet_{USED}$.

When the Link Layer of the master receives an LL_FEATURE_RSP PDU it shall only use procedures that are indicated in $featureSet_{USED}$.

An example of feature exchange is shown in Figure 5.2.

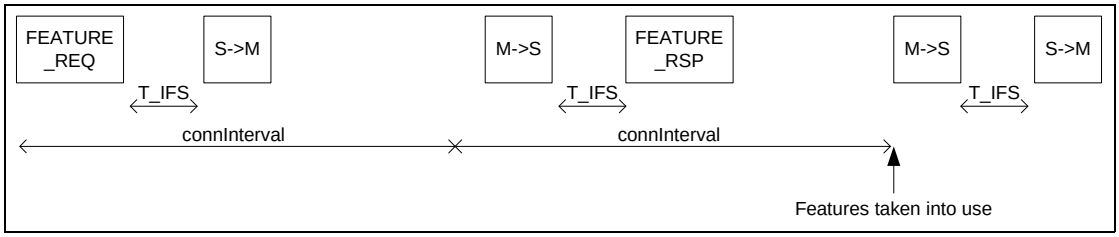



*Figure 5.2: Feature Exchange Procedure*

The procedure is complete in the master when the master receives the LL_FEATURE_RSP PDU from the slave.

### 5.1.5 Version Exchange

The Link Layer parameters for version information (*companyID*, *subVerNum*, *linkLayerVer,* as defined in Section 2.4.2.13) may be exchanged after entering the Connection State. Either the Link Layer of the master or slave can initiate this procedure by sending an LL_VERSION_IND PDU. This procedure should be used when requested by the Host. This procedure may be initiated autonomously by the Link Layer.

The Link Layer shall only queue for transmission a maximum of one LL_VERSION_IND PDU during a connection.

If the Link Layer receives an LL_VERSION_IND PDU and has not already sent an LL_VERSION_IND then the Link Layer shall send an LL_VERSION_IND PDU to the peer device.

If the Link Layer receives an LL_VERSION_IND PDU and has already sent an LL_VERSION_IND PDU then the Link Layer shall not send another LL_VERSION_IND PDU to the peer device.

The procedure has completed when an LL_VERSION_IND PDU has been received from the peer device.

### 5.1.6 Termination Procedure

This procedure is used for voluntary termination of a connection while in the Connection State. Voluntary termination occurs when the Host requests the Link Layer to terminate the connection. Either the Link Layer of the master or slave can initiate this procedure by sending an LL_TERMINATE_IND PDU. The termination procedure is not used in the event of the loss of the connection, for example after link supervision timeout or after a procedure timeout.

The Link Layer shall start a timer, $T_{terminate}$, when the LL_TERMINATE_IND PDU has been queued for transmission. The initiating Link Layer shall send LL_TERMINATE_IND PDUs until an acknowledgement is received or until the

timer, $T_{terminate}$, expires. The initial value for $T_{terminate}$ shall be set to value of the *connSupervisionTimeout*.

When the Link Layer receives an LL_TERMINATE_IND PDU it shall send the acknowledgement, exit the Connection State and shall transition to the Standby State.

The procedure has completed when the acknowledgement has been received.

## 5.2 PROCEDURE RESPONSE TIMEOUT

This section specifies procedure timeout rules that shall be applied to all the Link Layer control procedures specified in Section 5.1, except for the Connection Update and Channel Map Update procedures for which there are no timeout rules.

To be able to detect a non-responsive Link Layer Control Procedure, both the master and the slave shall use a procedure response timeout timer, $T_{PRT}$. Upon the initiation of a procedure, the procedure response timeout timer shall be reset and started.

Each LL Control PDU that is queued for transmission resets the procedure response timeout timer.

When the procedure completes, the procedure response timeout timer shall be stopped.

If the procedure response timeout timer reaches 40 seconds, the connection is considered lost. The Link Layer exits the Connection State and shall transition to the Standby State. The Host shall be notified of the loss of connection.

# SAMPLE DATA

***This part of the specification contains sample data for Bluetooth low energy. All sample data are provided for reference purpose only. They can be used to check the behavior of an implementation and avoid misunderstandings.***


Ford Ex. 1052

# CONTENTS

**1 Encryption sample data ..... 91**

1.1 Encrypt Command ..... 93

1.2 Derivation of the MIC and Encrypted Data ..... 93

# 1 ENCRYPTION SAMPLE DATA

This section contains sample data for the Low Energy encryption process.

The following scenario describes the start of encryption, followed by the transfer of an encrypted data channel data packet in each direction. It describes:

- how the derived values are calculated (fixed values are given in red)
- which HCI command and events are exchanged (given in italic)
- which LL messages are exchanged over the air (given in green).

Note: CRCs are not shown because they depend on a random CRC init value. Scrambling is disabled.

The following parameters are set to the fixed values below:

```
LTK = 0x4C68384139F574D836BCF34E9DFB01BF (MSO to LSO)
EDIV = 0x2474 (MSO to LSO)

RAND = 0xABCDEF1234567890 (MSO to LSO)
SKDm = 0xACBDCEDFE0F10213 (MSO to LSO)
SKDs = 0x0213243546576879 (MSO to LSO)
IVm = 0xBADCAB24 (MSO to LSO)
IVs = 0xDEAFBABE (MSO to LSO)
```

*HCI_LE_Start_Encryption (length 0x1C) – master HCI command*
*Pars (LSO to MSO) 00 08 90 78 56 34 12 ef cd ab 74 24 bf 01 fb 9d 4e f3 bc 36 d8 74 f5 39 41 38 68 4c*
*Handle (2-octet value MSO to LSO) 0x8000*
*Random (8-octet value MSO to LSO) 0xabcdef1234567890*
*Encrypted Diversifier (2-octet value MSO to LSO) 0x2474*
*Long Term Key (16-octet value MSO to LSO) 0x4c68384139f574d836bcf34e9dfb01bf*

**SKDm (LSO to MSO) :0x13:0x02:0xF1:0xE0:0xDF:0xCE:0xBD:0xAC:**

**IVm (LSO to MSO) :0x24:0xAB:0xDC:0xBA**

LL_ENC_REQ 03 17 03 90 78 56 34 12 ef cd ab 74 24 13 02 f1 e0 df ce bd ac 24 ab dc ba
Length 0x17
Control Type 0x03
Rand 90 78 56 34 12 ef cd ab
EDIV 74 24
SKDm 13 02 f1 e0 df ce bd ac
IVm 24 ab dc ba

**SKDs (LSO to MSO) :0x79:0x68:0x57:0x46:0x35:0x24:0x13:0x02:**

**IVs (LSO to MSO) :0xBE:0xBA:0xAF:0xDE**

*Sample Data*

```
LL_ENC_RSP 0b 0d 04 79 68 57 46 35 24 13 02 be ba af de
  Length 0x0D
  Control Type 0x04
  SKDs 79 68 57 46 35 24 13 02
  IVs be ba af de

IV = IVm || IVs
IV (LSO to MSO)      :0x24:0xAB:0xDC:0xBA:0xBE:0xBA:0xAF:0xDE

HCI_Long_Term_Key_Requested(length 0x0D) – slave event
  Pars (LSO to MSO)05 01 08 90 78 56 34 12 ef cd ab 74 24
    LE_Event_Code 0x05
    Handle (2-octet value MSO to LSO) 0x0801
    Random (8-octet value MSO to LSO) 0xabcdef1234567890
    Encrypted Diversifier (2-octet value MSO to LSO) 0x2474

HCI_LE_Long_Term_Key_Request_Reply (length 0x12) – slave command
  Pars (LSO to MSO)    01 08 bf 01 fb 9d 4e f3 bc 36 d8 74 f5 39 41 38 68 4c
    Handle (2-octet value MSO to LSO) 0x0801
    Key (16-octet value MSO to LSO) 0x4C68384139F574D836BCF34E9DFB01BF


SKD = SKDm || SKDs
SKD (LSO to MSO)
:0x13:0x02:0xF1:0xE0:0xDF:0xCE:0xBD:0xAC:0x79:0x68:0x57:0x46:0x35:0x24:0x13:0x02:

SK = Encrypt(LTK, SKD)
SK (LSO to MSO)
:0x66:0xC6:0xC2:0x27:0x8E:0x3B:0x8E:0x05:0x3E:0x7E:0xA3:0x26:0x52:0x1B:0xAD:0x99:

LL_START_ENC_REQ 07 01 05
  Length 0x0D
  Control Type 0x05

LL_START_ENC_RSP1 0f 05 9f cd a7 f4 48
  Length 0x05
  Control Type Encrypted:0x9F Clear:0x06
  MIC (32-bit value MSO to LSO) 0xCDA7F448 (note that MICs are sent MSO first on
the air)

LL_START_ENC_RSP2 07 05 a3 4c 13 a4 15
  Length 0x05
  Control Type Encrypted:0xA3 Clear:0x06
  MIC (32-bit value MSO to LSO) 0x4C13A415

HCI_ACL_Data_Packet Master host to controller
  00 08 1b 00 17 00 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f 70 71 31 32 33 34 35 36
37 38 39 30
    Handle (12-bit value MSO to LSO) 0x8000
    Data Total Length (16-bit value MSO to LSO) 0x001B (27 dec)
    Data (LSO to MSO) 17 00 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f 70 71 31 32 33
34 35 36 37 38 39 30

LL_DATA1 0e 1f 7a 70 d6 64 15 22 6d f2 6b 17 83 9a 06 04 05 59 6b d6 56 4f 79 6b 5b
9c e6 ff 32 f7 5a 6d 33
  Length 0x1F (i.e. 27 + 4 = 31 dec)
  Data (LSO to MSO)
```

```
    Clear     17 00 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f 70 71 31 32 33 34 35 36
37 38 39 30
    Encrypted 7a 70 d6 64 15 22 6d f2 6b 17 83 9a 06 04 05 59 6b d6 56 4f 79 6b 5b
9c e6 ff 32
  MIC (32-bit value MSO to LSO)   0xF75A6D33

HCI_ACL_Data_Packet Slave host to controller
  01 08 1b 00 17 00 37 36 35 34 33 32 31 30 41 42 43 44 45 46 47 48 49 4a 4b 4c 4d
4e 4f 50 51
    Handle (12-bit value MSO to LSO) 0x8001
    Data Total Length (16-bit value MSO to LSO) 0x001B (27 dec)
    Data (LSO to MSO) 17 00 37 36 35 34 33 32 31 30 41 42 43 44 45 46 47 48 49 4a
4b 4c 4d 4e 4f 50 51


LL_DATA2 06 1f f3 88 81 e7 bd 94 c9 c3 69 b9 a6 68 46 dd 47 86 aa 8c 39 ce 54 0d 0d
ae 3a dc df 89 b9 60 88
  Length 0x1F (i.e. 27 + 4 = 31 dec)
  Data (LSO to MSO)
    Clear     17 00 37 36 35 34 33 32 31 30 41 42 43 44 45 46 47 48 49 4a 4b 4c 4d
4e 4f 50 51
    Encrypted f3 88 81 e7 bd 94 c9 c3 69 b9 a6 68 46 dd 47 86 aa 8c 39 ce 54 0d 0d
ae 3a dc df
  MIC (32-bit value MSO to LSO)   0x89B96088
```

## 1.1 ENCRYPT COMMAND

```
HCI_LE_Encrypt (length 0x20) – command
  Pars (LSO to MSO) bf 01 fb 9d 4e f3 bc 36 d8 74 f5 39 41 38 68 4c 13 02 f1 e0 df
ce bd ac 79 68 57 46 35 24 13 02
    Key (16-octet value MSO to LSO):            0x4C68384139F574D836BCF34E9DFB01BF
    Plaintext_Data (16-octet value MSO to LSO): 0x0213243546576879acbdcedfe0f10213

HCI_Command_Complete (length 0x14) – event
  Pars (LSO to MSO) 02 17 20 00 66 c6 c2 27 8e 3b 8e 05 3e 7e a3 26 52 1b ad 99
    Num_HCI_Commands_Packets: 0x02
    Command_Opcode (2-octet value MSO to LSO): 0x2017
    Status: 0x00
    Encrypted_Data (16-octet value MSO to LSO): 0x99ad1b5226a37e3e058e3b8e27c2c666
```

## 1.2 DERIVATION OF THE MIC AND ENCRYPTED DATA

All B/X/A/S values below follow notation of AES-CCM: MSbyte to LSbyte & msbit to lsbit.

```
IV = DEAFBABEBADCAB24
SK = 99AD1B5226A37E3E058E3B8E27C2C666

1.START_ENC_RSP1 (packet 0, M --> S)
----------------

B0 = 49000000008024ABDCBABEBAAFDE0001
B1 = 00010300000000000000000000000000
B2 = 06000000000000000000000000000000
```

```
X1 = 712eaaaae60603521d245e50786eefe4
X2 = debc43782a022675fca0aa6f0854f1ab
X3 = 6399913fede5fa111bdb993bbfb9be06
=> MIC = 6399913f

A0 = 01000000008024ABDCBABEBAAFDE0000
A1 = 01000000008024ABDCBABEBAAFDE0001

S0 = ae3e6577f64a8f25408c9c10d53acf8e
S1 = 99190d88f4aa1b60b97ecfe6f5fee777

So, encrypted packet payload = 9F
    encrypted MIC = CDA7F448

Which results in the following packet:

LL_START_ENC_RSP1 - 0f 05 9f cd a7 f4 48
  Length: 05
  Control Type:
    Clear:     06
    Encrypted: 9f
  MIC: CD A7 F4 48


2.START_ENC_RSP2 (packet 0, S --> M)
----------------

B0 = 49000000000024ABDCBABEBAAFDE0001
B1 = 00010300000000000000000000000000
B2 = 06000000000000000000000000000000

X1 = ddc86e3094f0c29cf341ef4c2c1e0088
X2 = fe960f5c93fba45a53959842ea8a0c0a
X3 = db403db3a32f39156faf6a6b472e1010
=> MIC = db403db3

A0 = 01000000000024ABDCBABEBAAFDE0000
A1 = 01000000000024ABDCBABEBAAFDE0001

S0 = 975399a66acdc39124886930d7bca95f
S1 = a5add4127b2f43788ddc9cd86b0b89d2

So, encrypted packet payload = A3
    encrypted MIC = 4c13a415

Which results in the following packet:

LL_START_ENC_RSP2 07 05 a3 4c 13 a4 15
  Length: 05
  Control Type:
```

```
    Clear:     06
    Encrypted: A3
  MIC: 4c 13 a4 15


3. Data packet1 (packet 1, M --> S)
---------------

B0 = 49010000008024ABDCBABEBAAFDE001B
B1 = 00010200000000000000000000000000
B2 = 1700636465666768696A6B6C6D6E6F70
B3 = 71313233343536373839300000000000

X1 = 7c688612996de101f3eacb68b443969c
X2 = e3f1ef5c30161c0a9ec07274a0757fc8
X3 = e7e346f5b7c8a6072890a60dcf4ec20a
X4 = 3db113320b182f9fed635db14cac2df0
=> MIC = 3db11332

A0 = 01010000008024ABDCBABEBAAFDE0000
A1 = 01010000008024ABDCBABEBAAFDE0001
A2 = 01010000008024ABDCBABEBAAFDE0002

S0 = caeb7e017296dd2fa9a2ce789179501a
S1 = 6d70b50070440a9a027de8f66b6a6a29
S2 = 1ae7647c4d5e6dabdec602404c302341


So, encrypted packet payload =

7A70D66415226DF26B17839A060405596BD6564F796B5B9CE6FF32
    encrypted MIC = F75A6D33


which results in the following packet:


LL_DATA1 0E 1F 7A 70 D6 64 15 22 6D F2 6B 17 83 9A 06 04 05 59 6B D6
56 4F 79 6B 5B 9C E6 FF 32 F7 5A 6D 33
  Length: 1F
  Data:
    Clear:     17 00 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f 70 71 31
32 33 34 35 36 37 38 39 30
    Encrypted: 7A 70 D6 64 15 22 6D F2 6B 17 83 9A 06 04 05 59 6B D6
56 4F 79 6B 5B 9C E6 FF 32
  MIC: F7 5A 6D 33

4. Data packet2 (packet 1, S --> M)
---------------

B0 = 49010000000024ABDCBABEBAAFDE001B
B1 = 00010200000000000000000000000000
B2 = 17003736353433323130414243444546
```

*Sample Data*

```
B3 = 4748494A4B4C4D4E4F50510000000000

X1 = 714234d50d6f1da5663be3e78460ad87
X2 = 96df1d97959e6176ac215c7baf90c674
X3 = 6cc52c3dcecdc2fa81eb347887960673
X4 = a776a26be617366496c391e36f6374a1 => MIC = a776a26b

A0 = 01010000000024ABDCBABEBAAFDE0000
A1 = 01010000000024ABDCBABEBAAFDE0001
A2 = 01010000000024ABDCBABEBAAFDE0002

S0 = 2ecfc2e31e01875653c0f306fc7bfb96
S1 = e488b6d188a0faf15889e72a059902c0
S2 = edc470841f4140e0758c8e8f708399bd


So, encrypted packet payload =

F38881E7BD94C9C369B9A66846DD4786AA8C39CE540D0DAE3ADCDF
    encrypted MIC = 89B96088


Which results in the following packet:


LL_DATA2 06 1F F3 88 81 E7 BD 94 C9 C3 69 B9 A6 68 46 DD 47 86 AA 8C
39 CE 54 0D 0D AE 3A DC DF 89 B9 60 88
  Length: 1F
  Data:
    Clear:     17 00 37 36 35 34 33 32 31 30 41 42 43 44 45 46 47 48
49 4a 4b 4c 4d 4e 4f 50 51
    Encrypted: F3 88 81 E7 BD 94 C9 C3 69 B9 A6 68 46 DD 47 86 AA 8C
39 CE 54 0D 0D AE 3A DC DF

MIC: 89 B9 60 88
```

# MESSAGE SEQUENCE CHARTS

***Examples of message sequence charts showing the interactions of the Host Controller Interface with the Link Layer.***


Ford Ex. 1052

# CONTENTS

**1 Introduction ..... 100**
1.1 Notation ..... 100
1.2 Control Flow ..... 101
1.3 Example MSC ..... 101
**2 Standby State ..... 102**
2.1 Initial Setup ..... 102
2.2 Random Device Address ..... 103
2.3 White Lists ..... 103
**3 Advertising State ..... 104**
3.1 Undirected Advertising ..... 104
3.2 Directed Advertising ..... 105
**4 Scanning State ..... 106**
4.1 Passive Scanning ..... 106
4.2 Active Scanning ..... 106
**5 Initiating State ..... 108**
5.1 Initiating a Connection ..... 108
5.2 Canceling an Initiation ..... 108
**6 Connection State ..... 110**
6.1 Sending Data ..... 110
6.2 Connection Update ..... 110
6.3 Channel Map Update ..... 111
6.4 Features Exchange ..... 111
6.5 Version Exchange ..... 112
6.6 Start Encryption ..... 113
6.7 Start Encryption without Long Term Key ..... 114
6.8 Start Encryption with Event Masked ..... 115
6.9 Start Encryption Without Slave Supporting Encryption ..... 115
6.10 Restart Encryption ..... 116
6.11 Disconnect ..... 116

# 1 INTRODUCTION

This section shows typical interactions between Host Controller Interface (HCI) Commands and Events and the Link Layer (LL). It focuses on the message sequence charts (MSCs) for the procedures specified in “Bluetooth Host Controller Interface Functional Specification” with regard to Link Layer Control Procedures from “Link Layer”. This section illustrates only the most useful scenarios; it does not cover all possible alternatives. Furthermore, the message sequence charts do not consider errors over the air interface or host interface. In all message sequence charts it is assumed that all events are not masked, so the Host Controller will not filter out any events.

The sequence of messages in these message sequence charts is for illustrative purposes. The messages may be sent in a different order where allowed by the Link Layer or HCI sections. If any of these charts differ with text in the Link Layer or HCI sections, the text in those sections shall be considered normative. This section is informative.

## 1.1 NOTATION

The notation used in the message sequence charts (MSCs) consists of ovals, elongated hexagons, boxes, lines, and arrows. The vertical lines terminated on the top by a shadow box and at the bottom by solid oval indicate a protocol entity that resides in a device. MSCs describe interactions between these entities and states those entities may be in.

The following symbols represent interactions and states:

| **Oval** | **Defines the context for the message sequence chart** |
|---|---|
| Hexagon | Indicates a condition needed to start the transactions below this hexagon. The location and width of the Hexagon indicates which entity or entities make this decision. |
| Box | Replaces a group of transactions. May indicate a user action, or a procedure in the baseband. |
| Dashed Box | Optional group of transactions. |
| Solid Arrow | Represents a message, signal or transaction. Can be used to show Link Layer and HCI traffic.<br>Some baseband packet traffic is also shown. These are prefixed by BB followed by either the type of packet, or an indication that there is an ACK signal in a packet. |
| Dashed Arrow | Represents a optional message, signal or transaction. Can be used to show Link Layer and HCI traffic. |

## 1.2 CONTROL FLOW

Some message sequences are split into several charts. These charts are marked in sequence with different step numbers with multiple paths through with optional letters after the step numbers. Numbers indicate normal or required ordering. The letters represent alternative paths. For example, Step 4 is after Step 3, and Step 5a could be executed instead of Step 5b.

## 1.3 EXAMPLE MSC

The protocol entities represented in the example shown in Figure 1.1 illustrate the interactions of two devices named A and B. Note that each device includes a Host and a LL entity in this example. Other MSCs in this section may show the interactions of more than two devices.

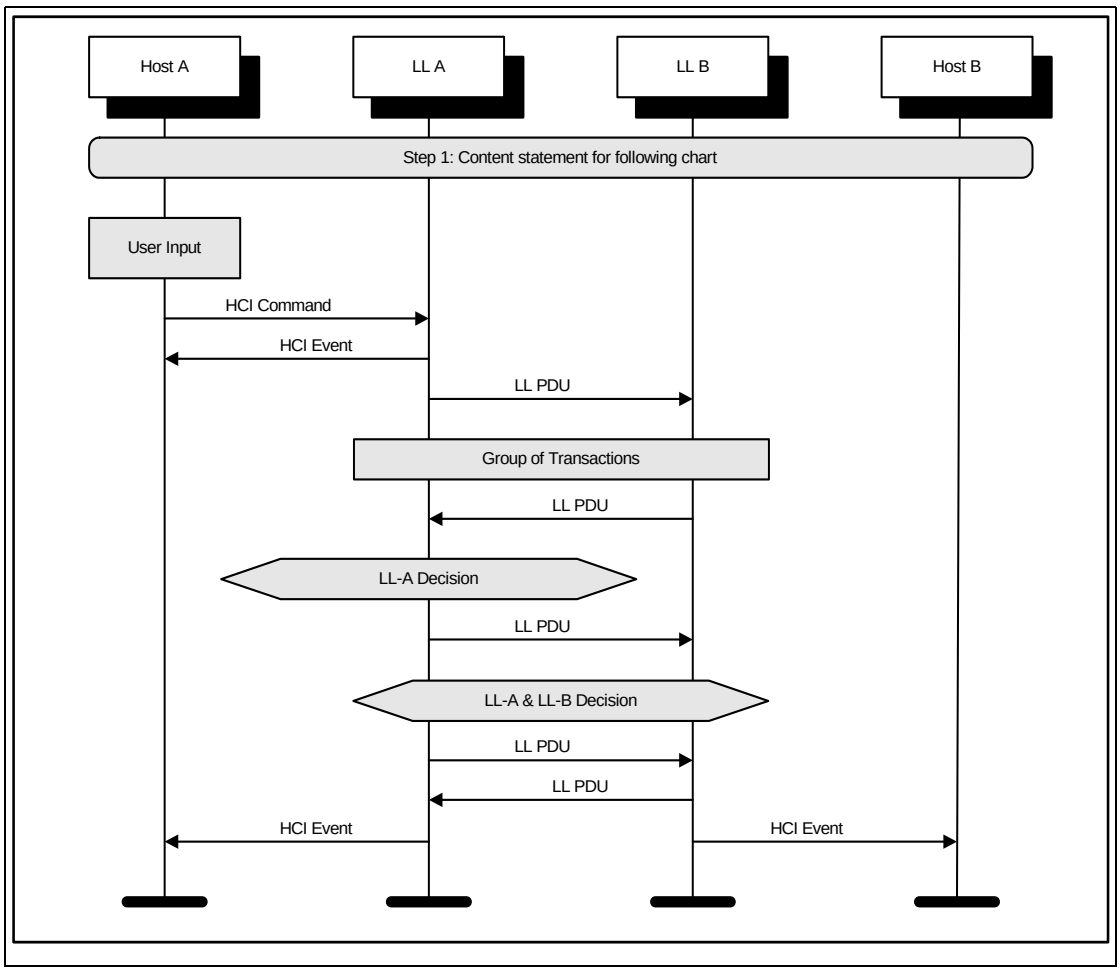



*Figure 1.1: Example MSC*

# 2 STANDBY STATE

## 2.1 INITIAL SETUP

To perform initial setup of a LE Controller, the following sequence of actions may be required.

First, the host would wait for the Controller to indicate the number of HCI Command Packets the Host is currently allowed to send using a Command Complete event on a No OPeration command opcode. Then it would reset the Controller to a known state. Then it needs to read the local supported features to check that low energy is supported on this Controller. It would then set the event mask and LE event mask to enable the events that it wants the Controller to generate to the Host. Next, it will check the buffers that are available for data flow, using the Read Buffer Size and LE Read Buffer Size commands. Then it would read the locally supported LE features and select the features that it wishes to use. Finally, it will read the public device address if the Controller has one (see Figure 2.1).

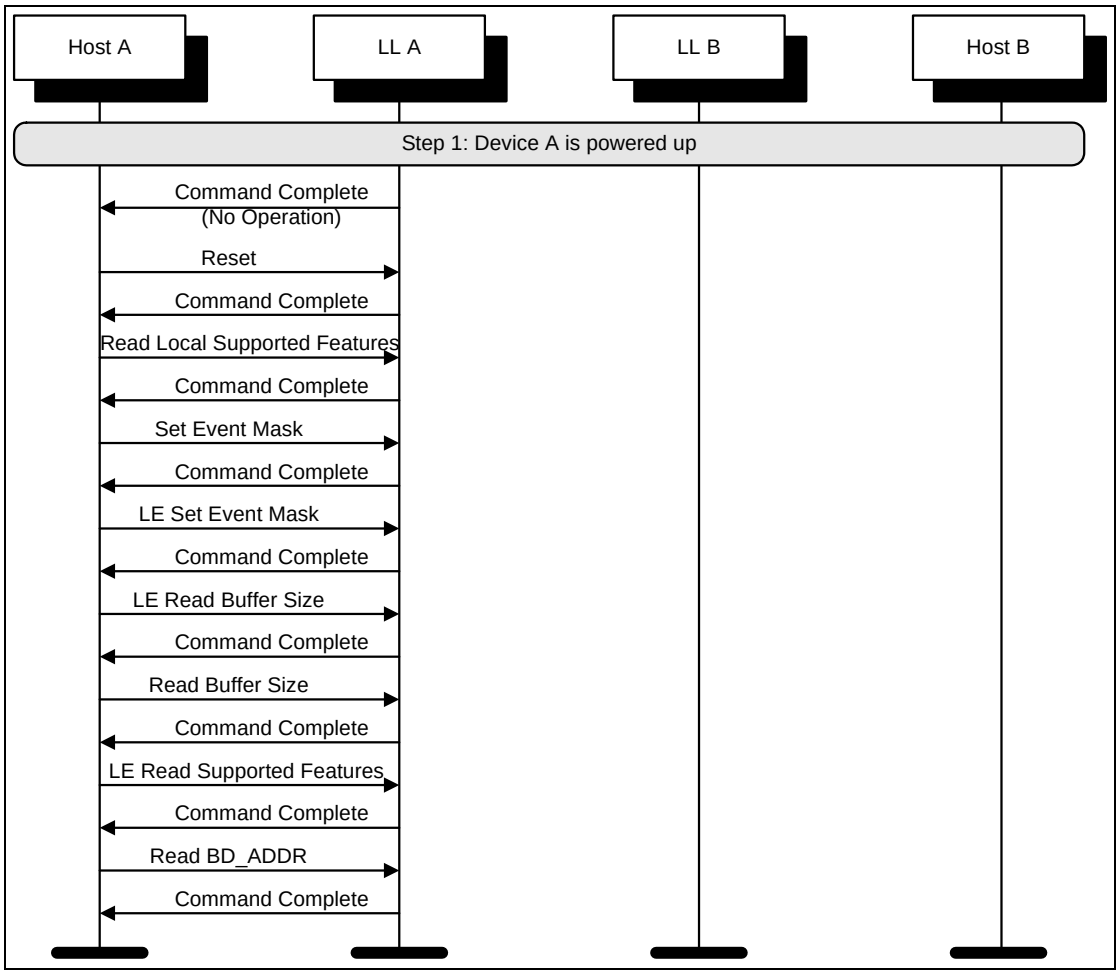



*Figure 2.1: Initial Setup*

## 2.2 RANDOM DEVICE ADDRESS

A device may use a random device address, but this address has to be configured before being used during advertising, scanning or initiating (see Figure 2.2).

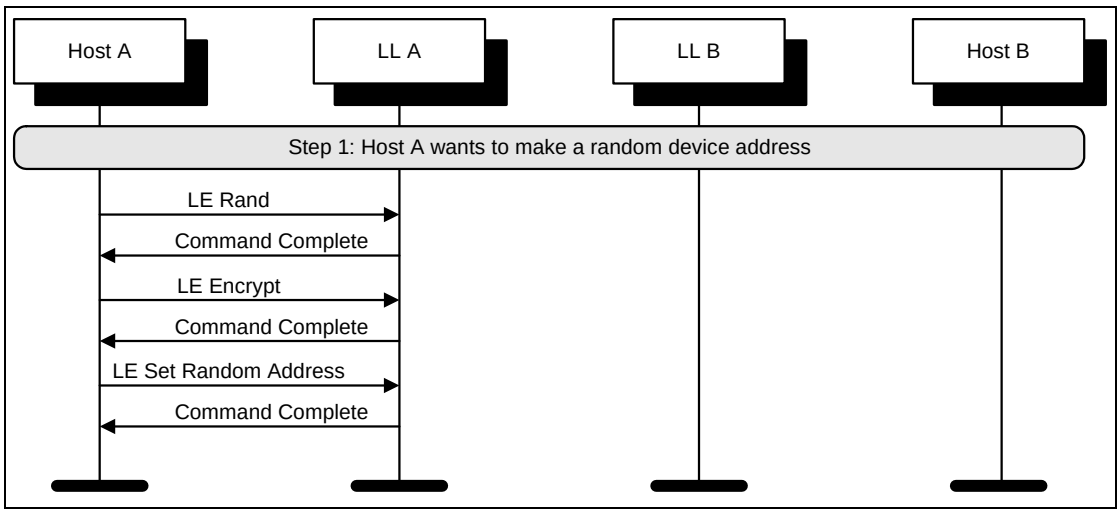



*Figure 2.2: Random Device Address*

## 2.3 WHITE LISTS

Before advertising, scanning or initiating can use white lists, the white list may be cleared and devices added in as required (see Figure 2.3).

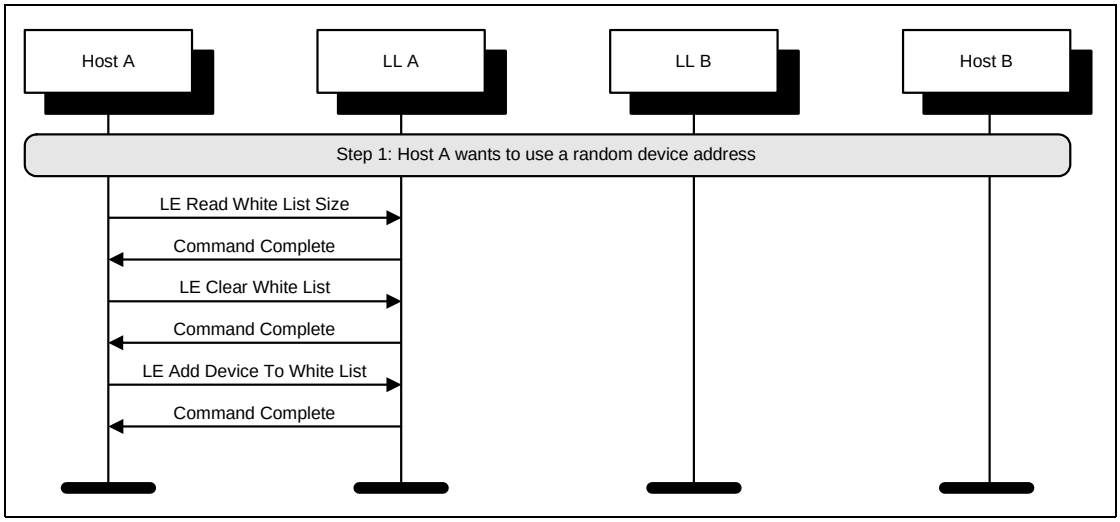



*Figure 2.3: White Lists*

# 3 ADVERTISING STATE

## 3.1 UNDIRECTED ADVERTISING

A device may enter the Advertising State by enabling advertising. It should also configure the advertising parameters before doing this (see Figure 3.1).

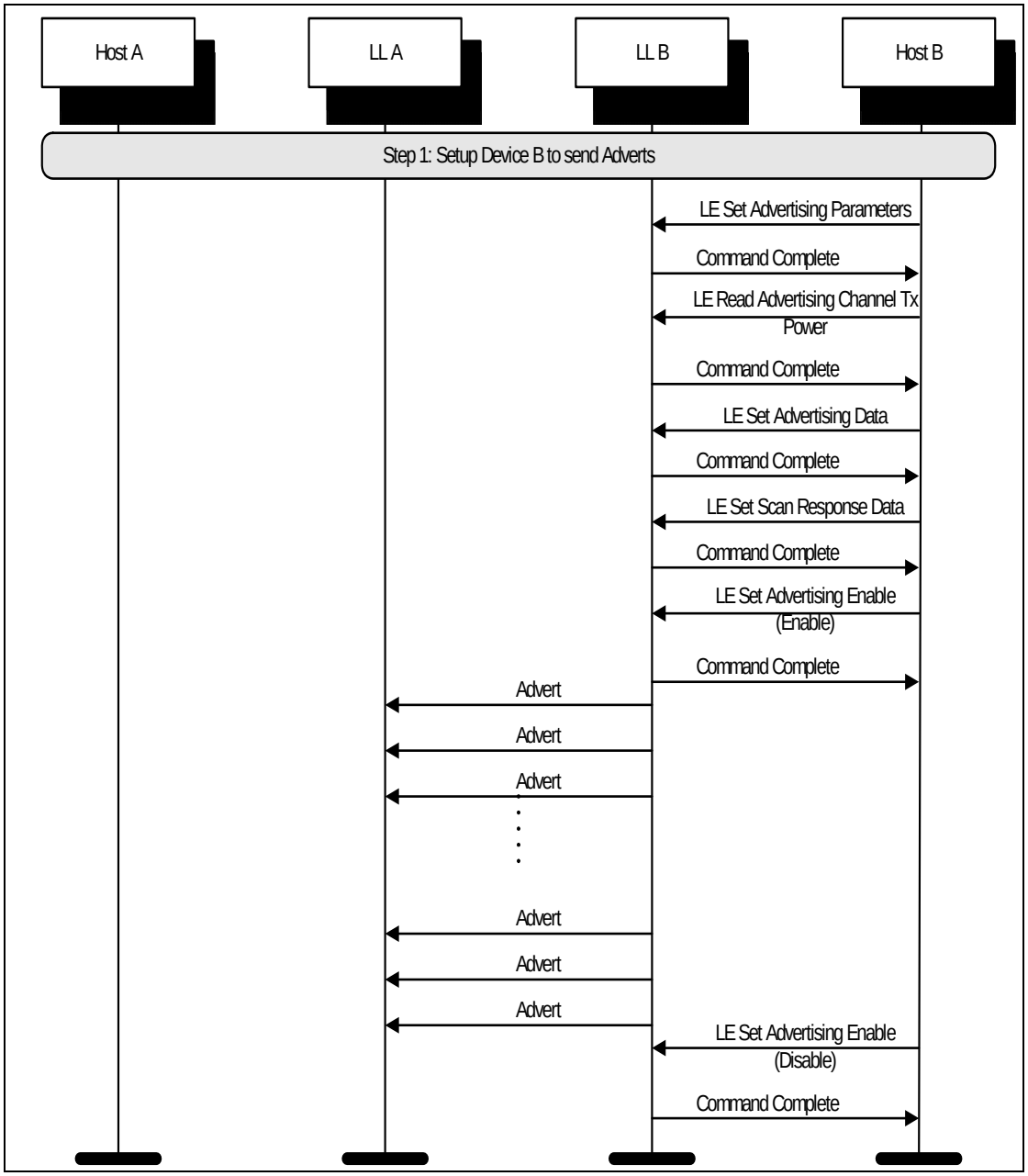



*Figure 3.1: Undirected Advertising*

## 3.2 DIRECTED ADVERTISING

A device may use directed advertising to allow an initiator to connect to it. Directed advertising is time limited in the Controller and therefore this may fail before a connection is created. This example only shows the failure case (see Figure 3.2).

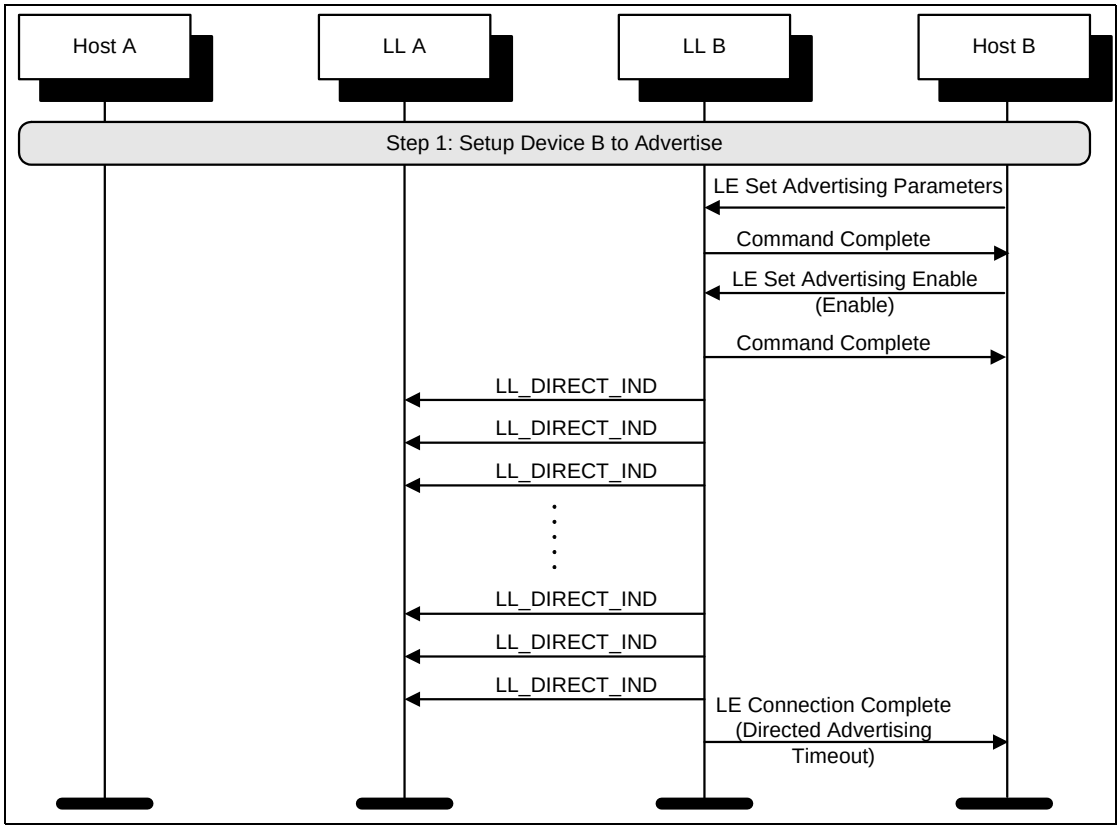



*Figure 3.2: Directed Advertising showing failure case*

# 4 SCANNING STATE

## 4.1 PASSIVE SCANNING

A device can use passive scanning to find advertising devices in the area. This would receive advertising packets from peer devices and report these to the Host (see Figure 4.1).

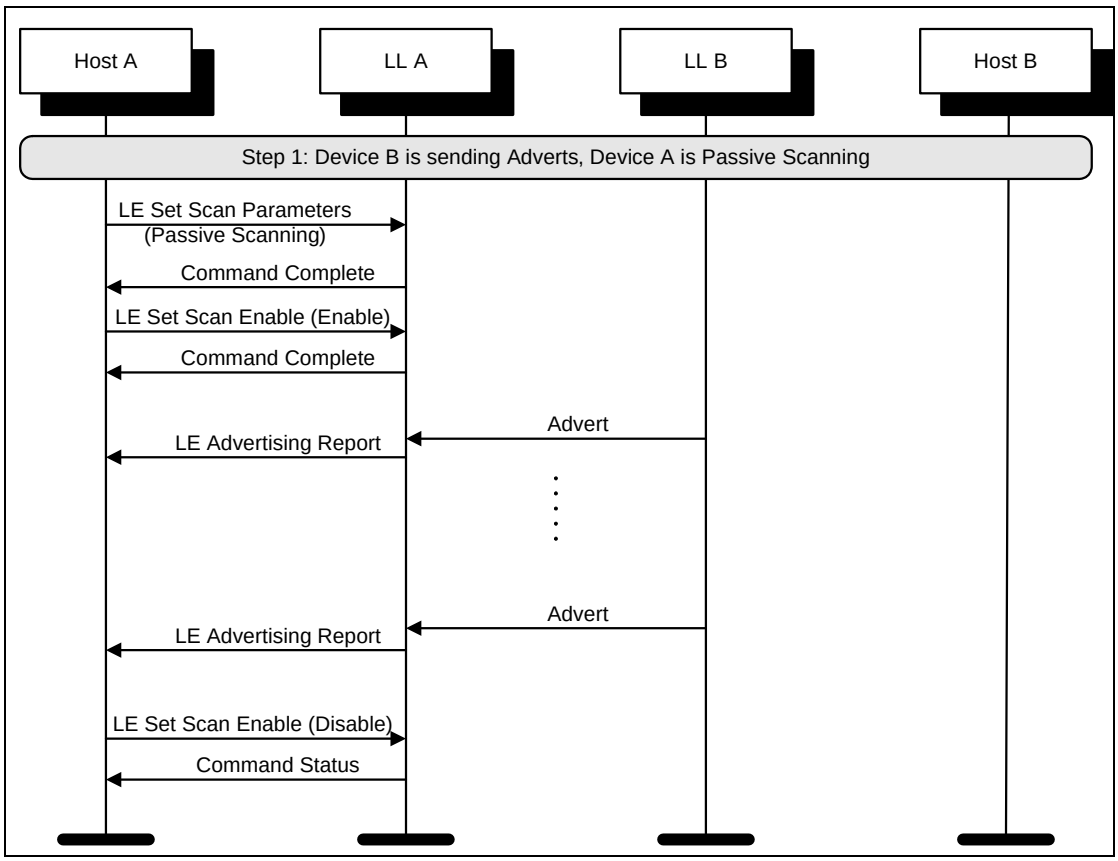



*Figure 4.1: Passive Scanning*

## 4.2 ACTIVE SCANNING

A device may use active scanning to obtain more information about devices that may be useful to populate a user interface. Active scanning involves more link layer advertising messages (see Figure 4.2).

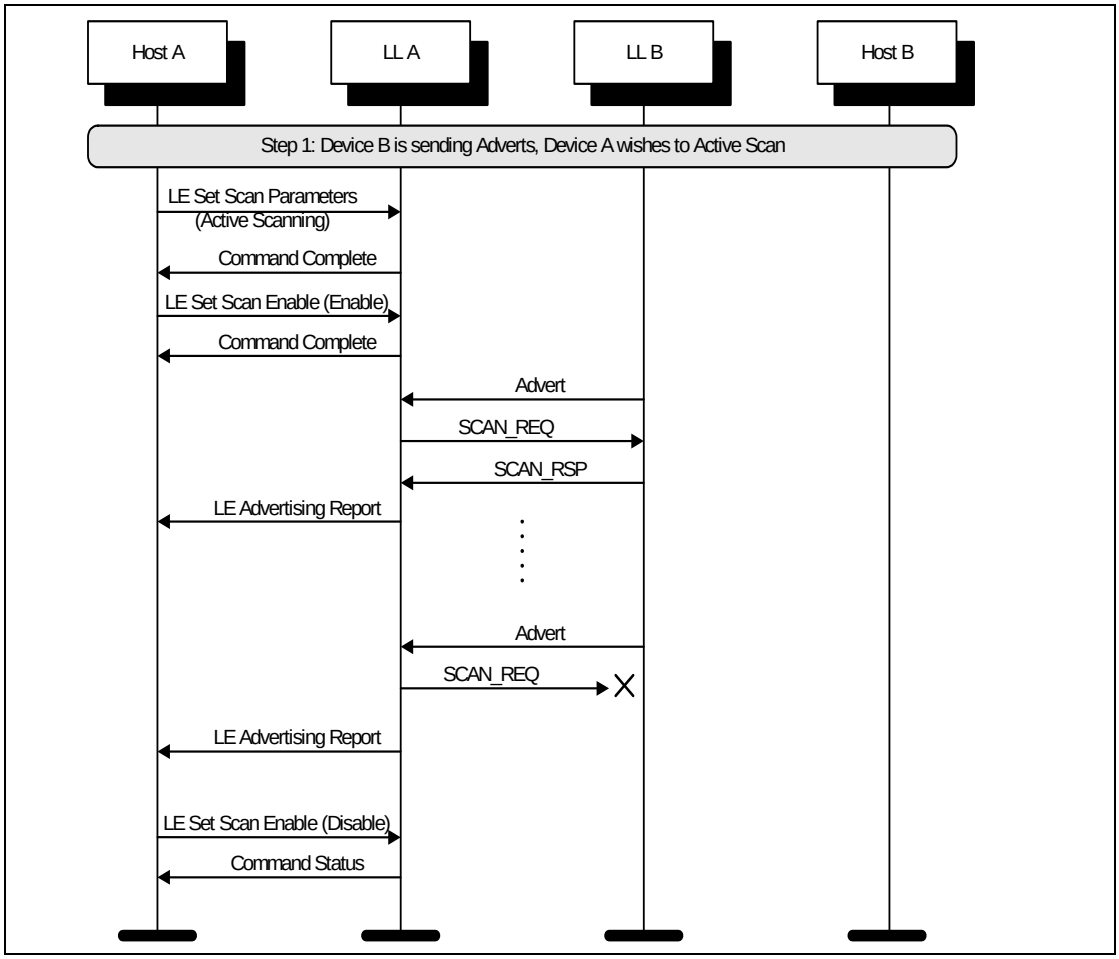



*Figure 4.2: Active Scanning*

# 5 INITIATING STATE

## 5.1 INITIATING A CONNECTION

A device can initiate a connection to an advertiser. This example shows a successful initiation, resulting in both devices able to send application data (see Figure 5.1).

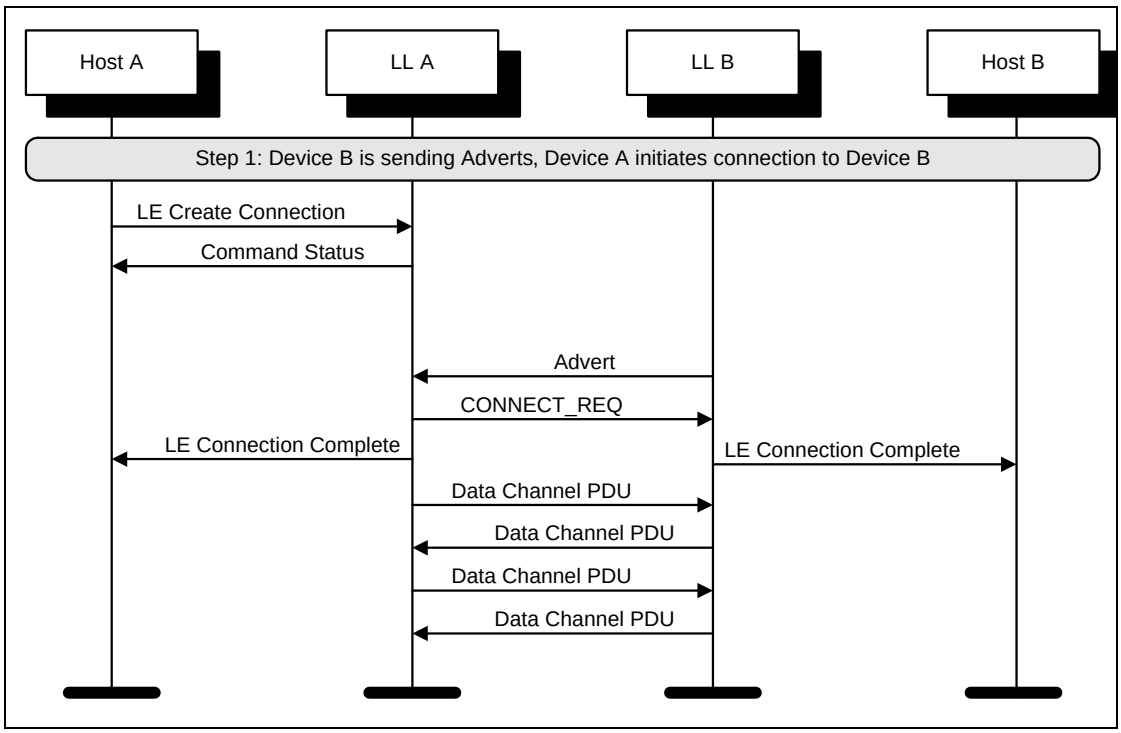



*Figure 5.1: Initiating a Connection*

## 5.2 CANCELING AN INITIATION

A device can cancel a pending connection creation. This example shows an unsuccessful initiation, followed by a cancellation of the initiation (see Figure 5.2).

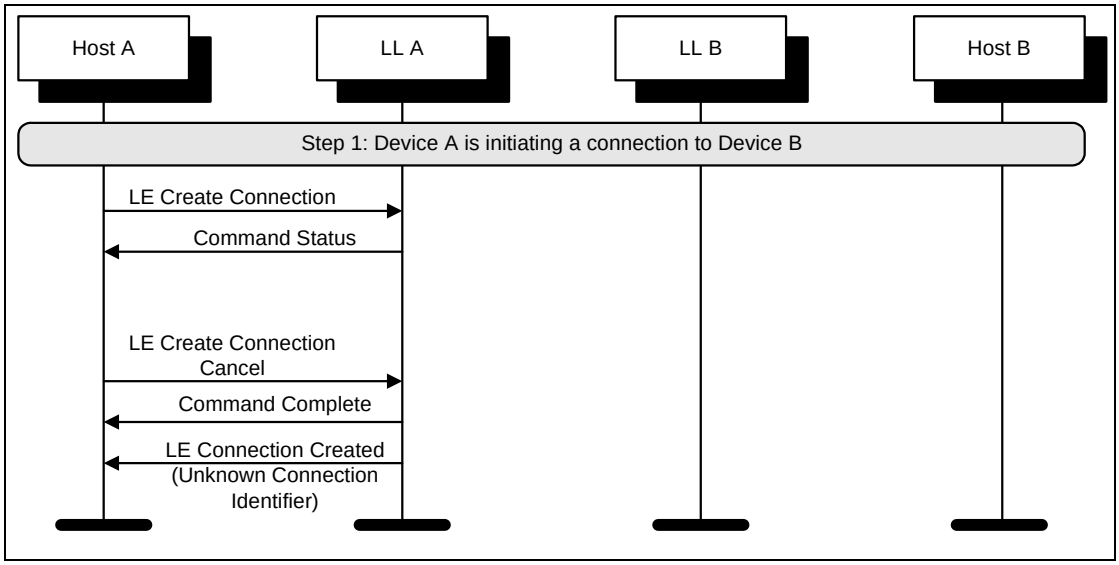



*Figure 5.2: Canceling an Initiation*

# 6 CONNECTION STATE

## 6.1 SENDING DATA

Once two devices are in a connection, either device can send data. This example shows both devices sending data, for example when the attribute protocol does a read request and a read response is returned (see Figure 6.1).

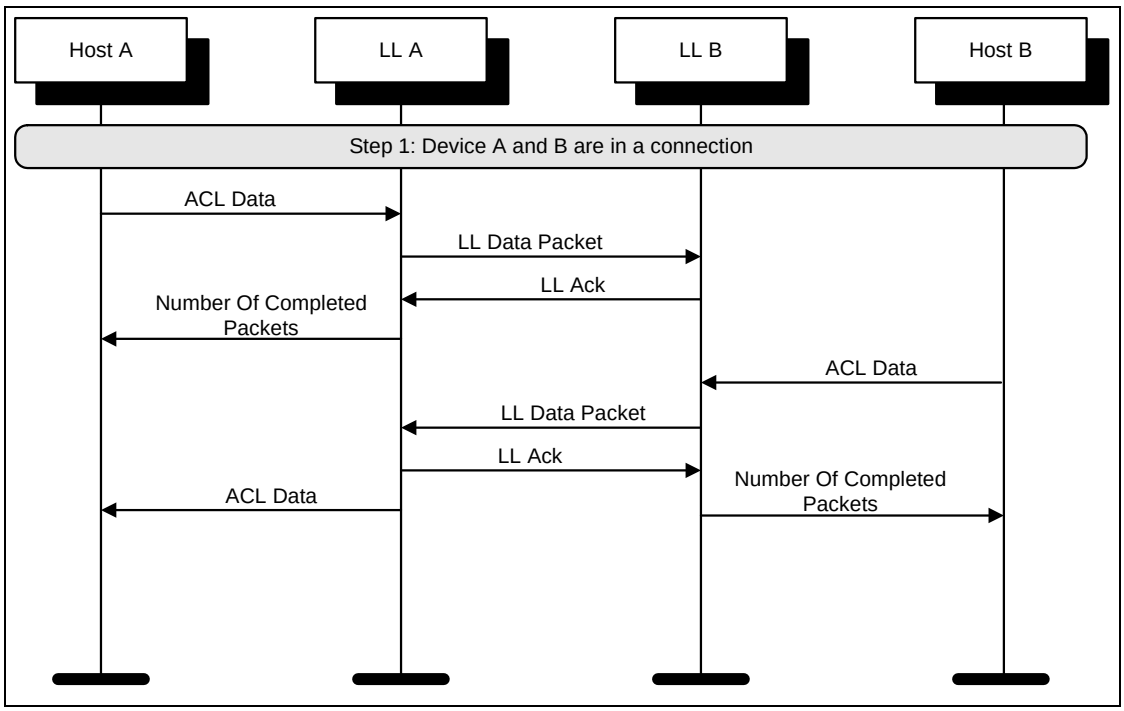



*Figure 6.1: Sending Data*

## 6.2 CONNECTION UPDATE

The master of the connection may request a connection update using a Link Layer Control Procedure (see Figure 6.2).

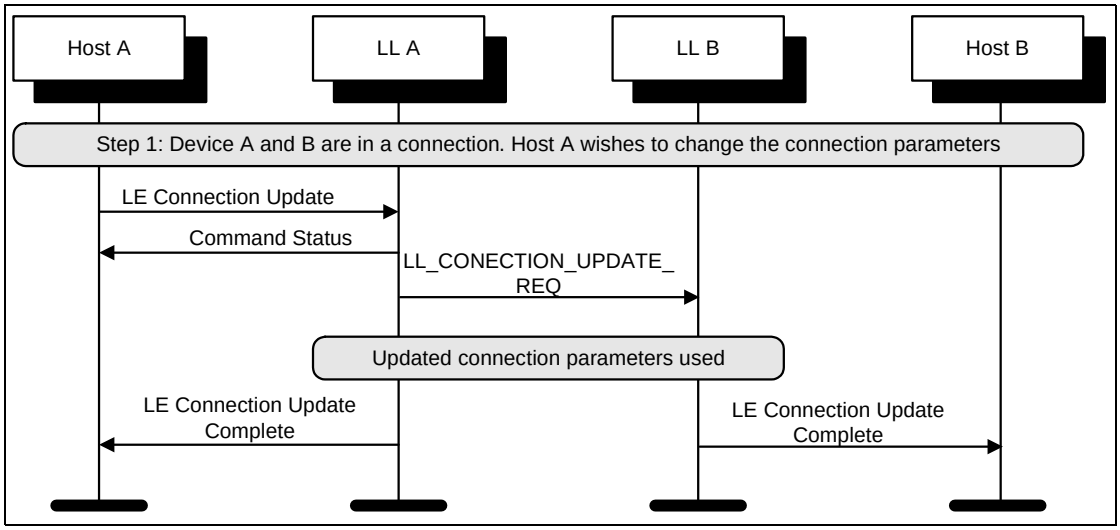



*Figure 6.2: Connection Update*

## 6.3 CHANNEL MAP UPDATE

The Controller of the master may receive some channel classification data from the Host and then perform the Channel Update Link Layer Control Procedure (see Figure 6.3).

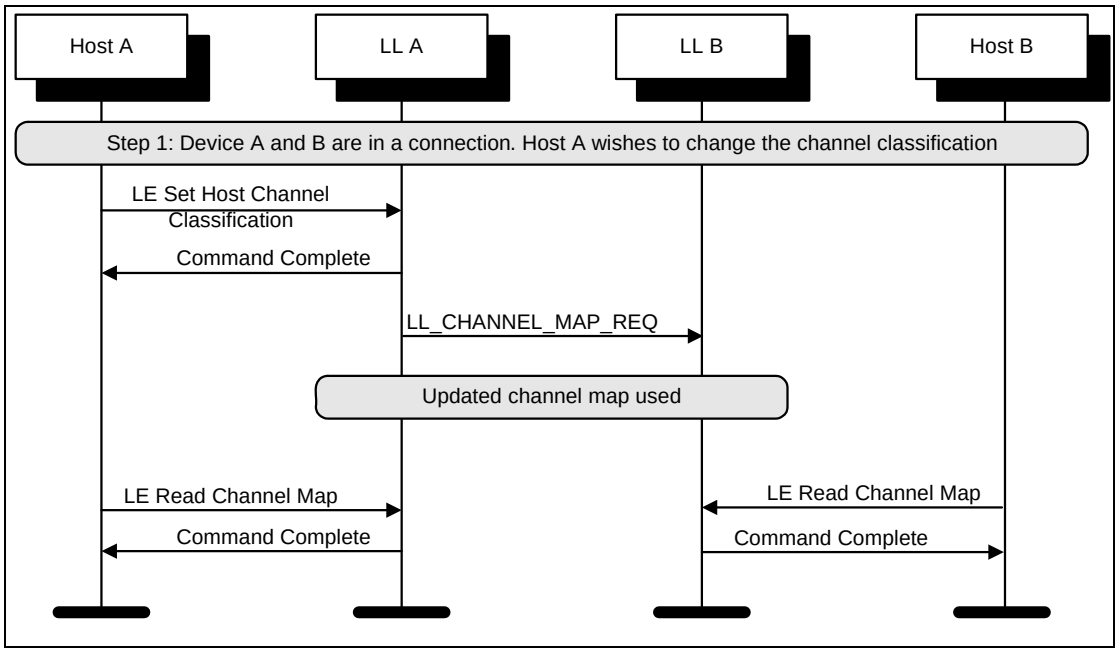



*Figure 6.3: Channel Map Update*

## 6.4 FEATURES EXCHANGE

The master device can discover the set of features available on the slave device. To achieve this, the Feature Exchange Link Layer Control Procedure is used (see Figure 6.4).

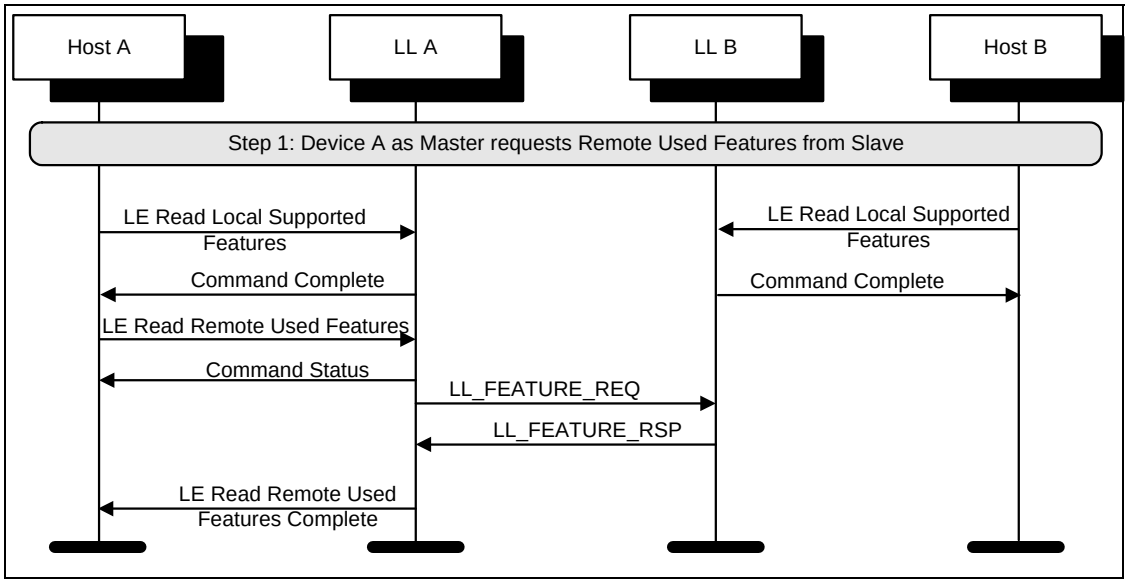



*Figure 6.4: Features Exchange*

## 6.5 VERSION EXCHANGE

Either device may perform a version exchange (see Figure 6.5 and Figure 6.6).

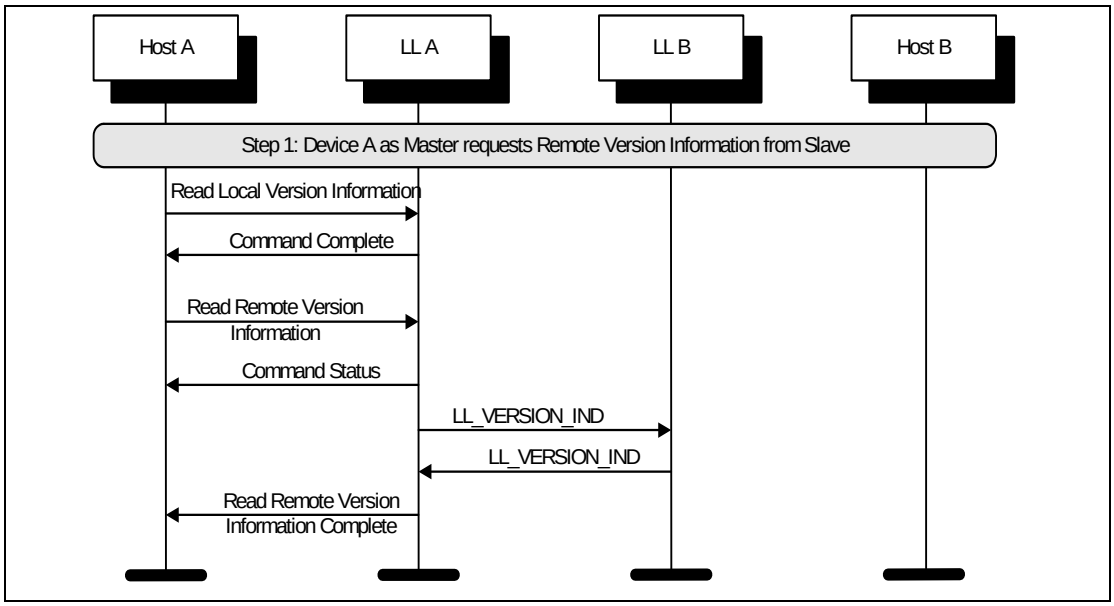



*Figure 6.5: Version Exchange from Master*

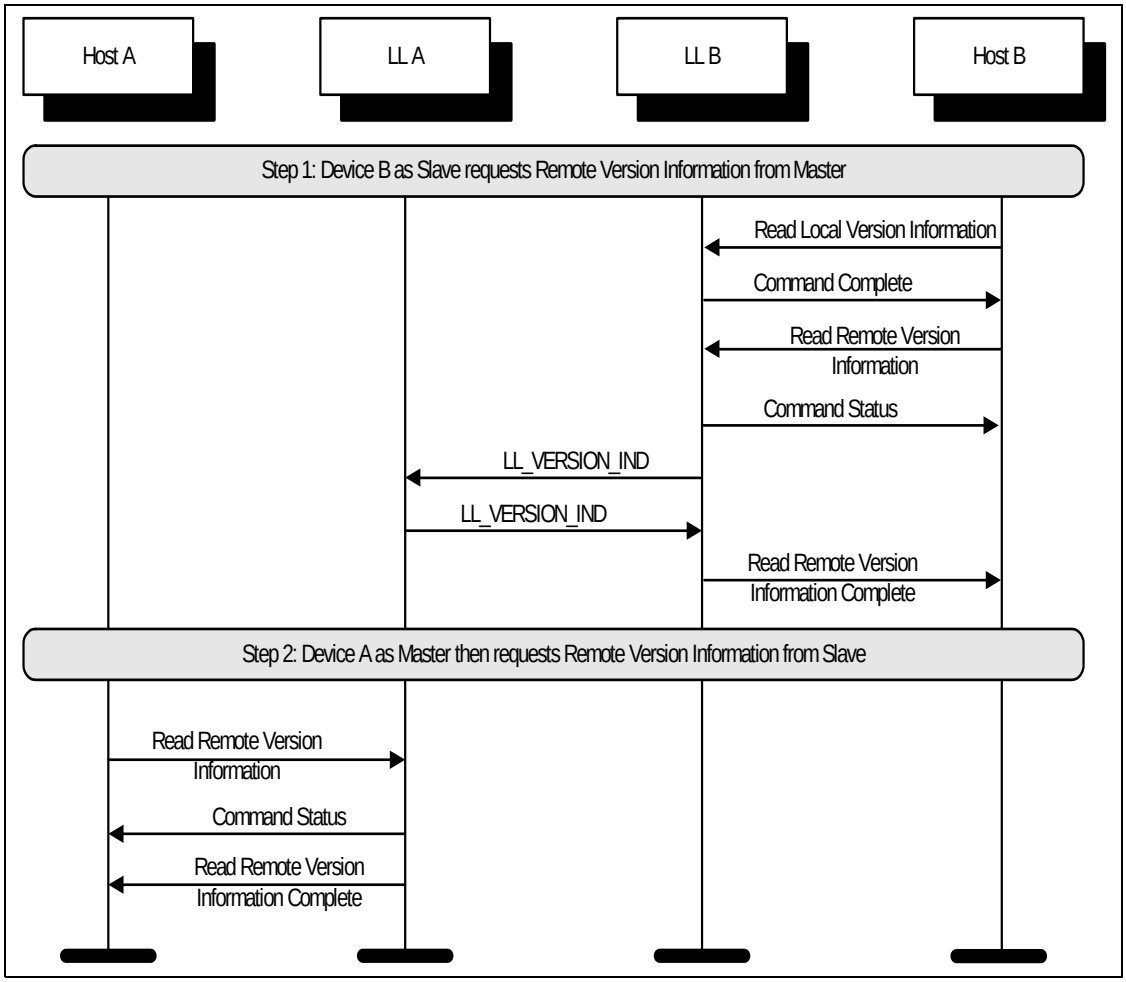



*Figure 6.6: Version Exchange from Slave*

## 6.6 START ENCRYPTION

If encryption has not been started on a connection, it may be started by the master (see Figure 6.7).

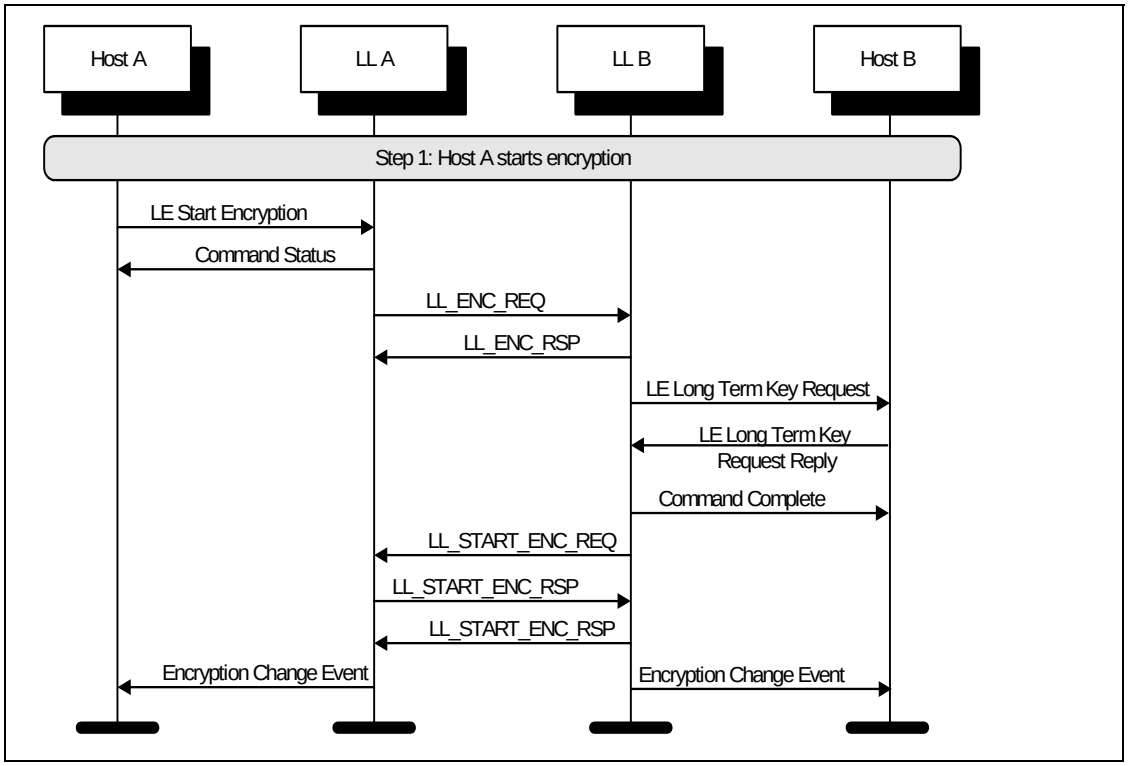



*Figure 6.7: Start Encryption*

## 6.7 START ENCRYPTION WITHOUT LONG TERM KEY

If encryption has not been started on a connection, it may be started by the master. Figure 6.8 shows the failure case of the slave not having the long term key for the master.

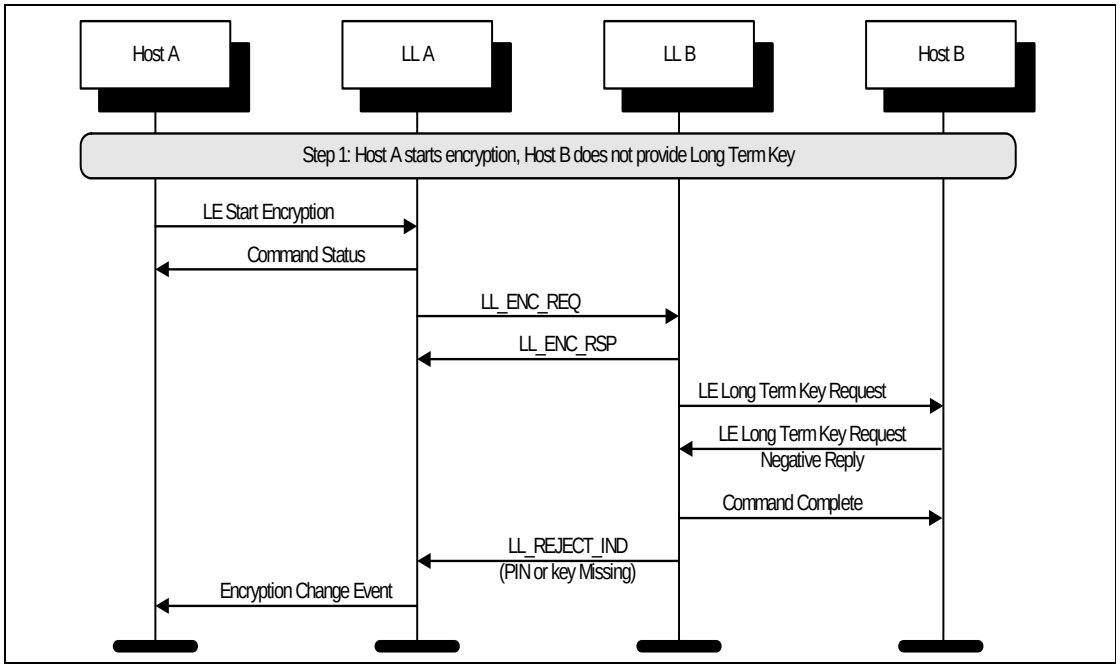



*Figure 6.8: Start encryption without long-term key*

## 6.8 START ENCRYPTION WITH EVENT MASKED

If encryption has not been started on a connection, it may be started by the master. Figure 6.9 shows the failure case when the slave has masked out the LE Long Term Key Request event.

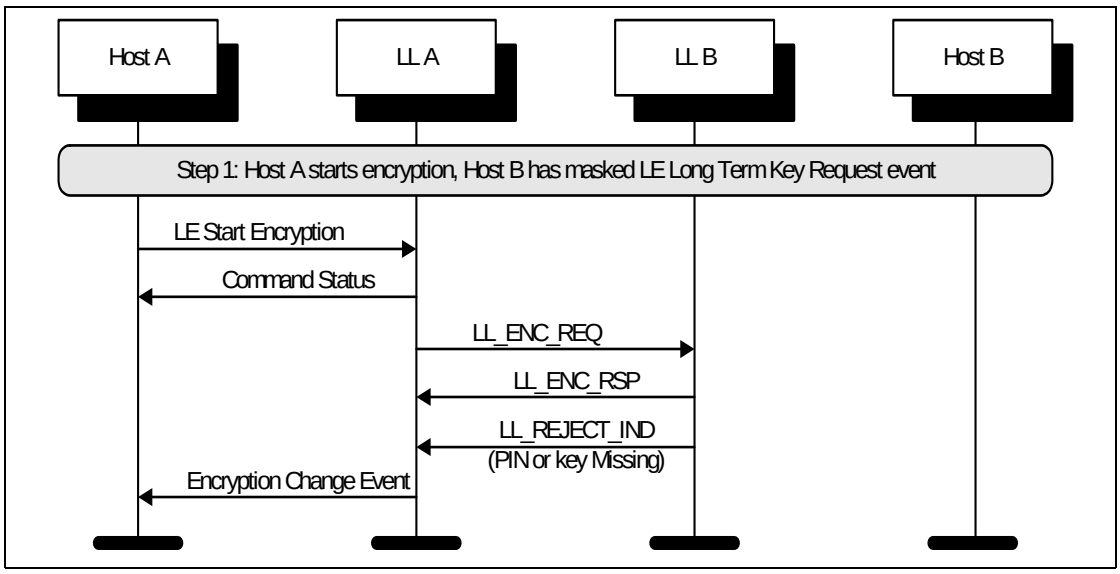



*Figure 6.9: Start encryption with slave masking out event*

## 6.9 START ENCRYPTION WITHOUT SLAVE SUPPORTING ENCRYPTION

If Encryption has not been started on a connection, it may be started by the master. Figure 6.10 shows the failure case of the slave that does not support the encryption feature.

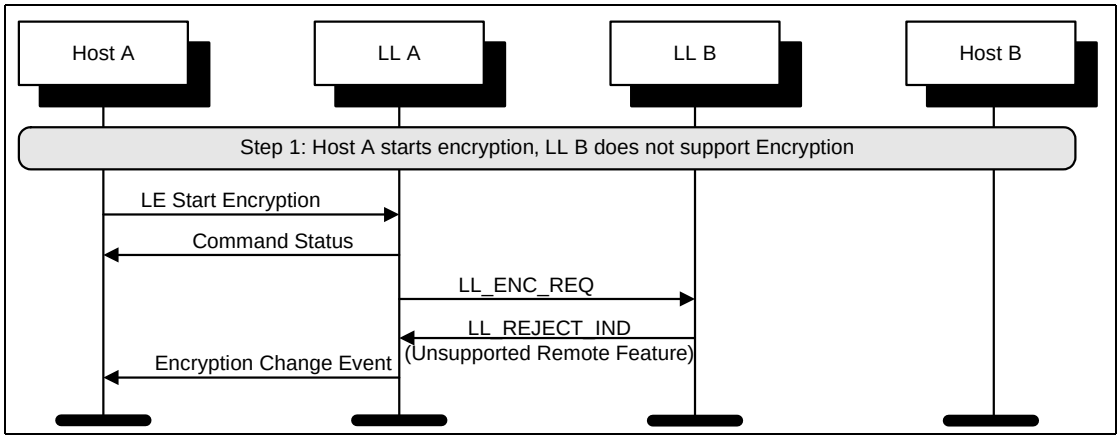



*Figure 6.10: Start Encryption failure when slave does not support encryption*

## 6.10 RESTART ENCRYPTION

If encryption has already been started on a connection, it may be restarted by the master. This may be required to use a stronger encryption as negotiated by the Security Manager Protocol (see Figure 6.11).

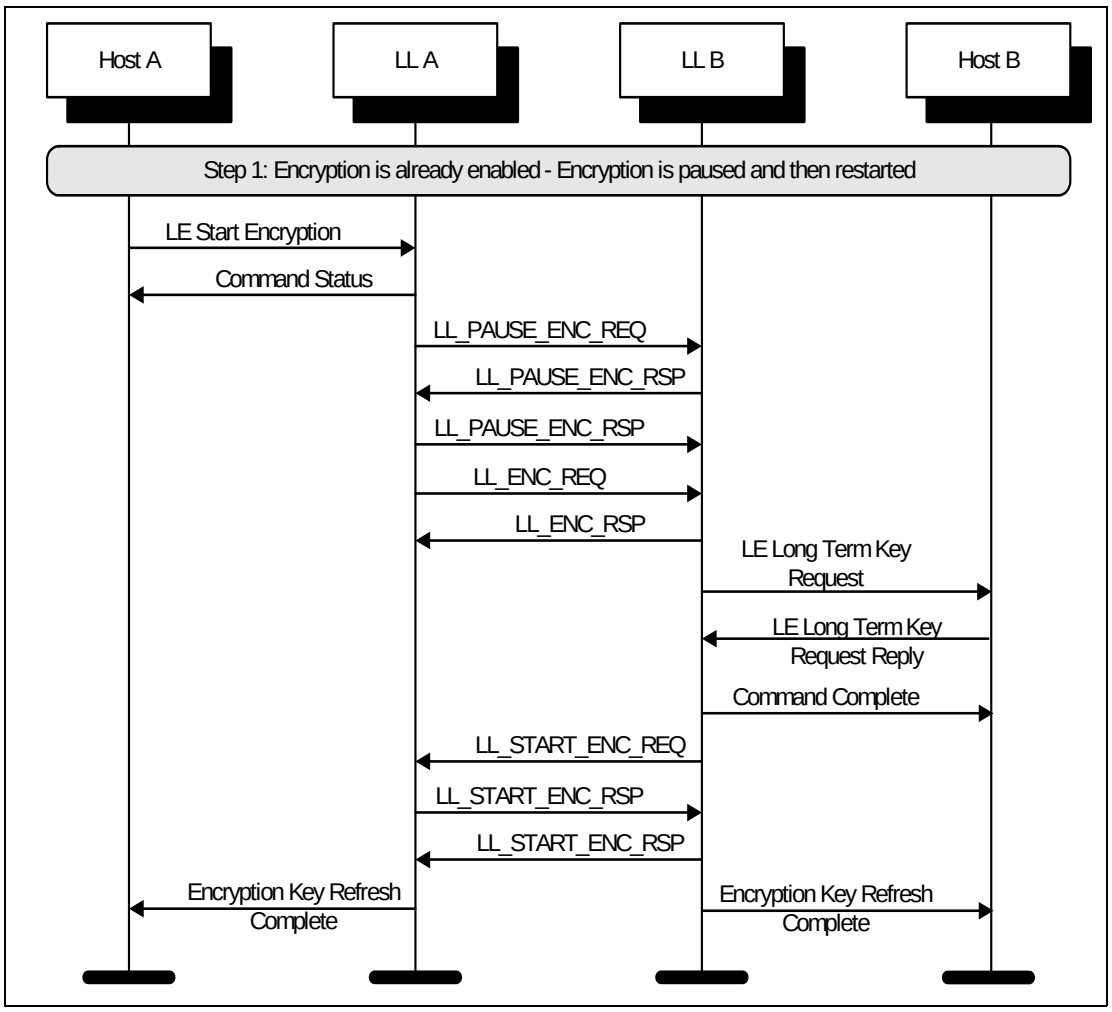



*Figure 6.11: Restart Encryption*

## 6.11 DISCONNECT

Once a connection has no need to be kept active, the Host can disconnect it. This can be done by either device (see Figure 6.12 and Figure 6.13).

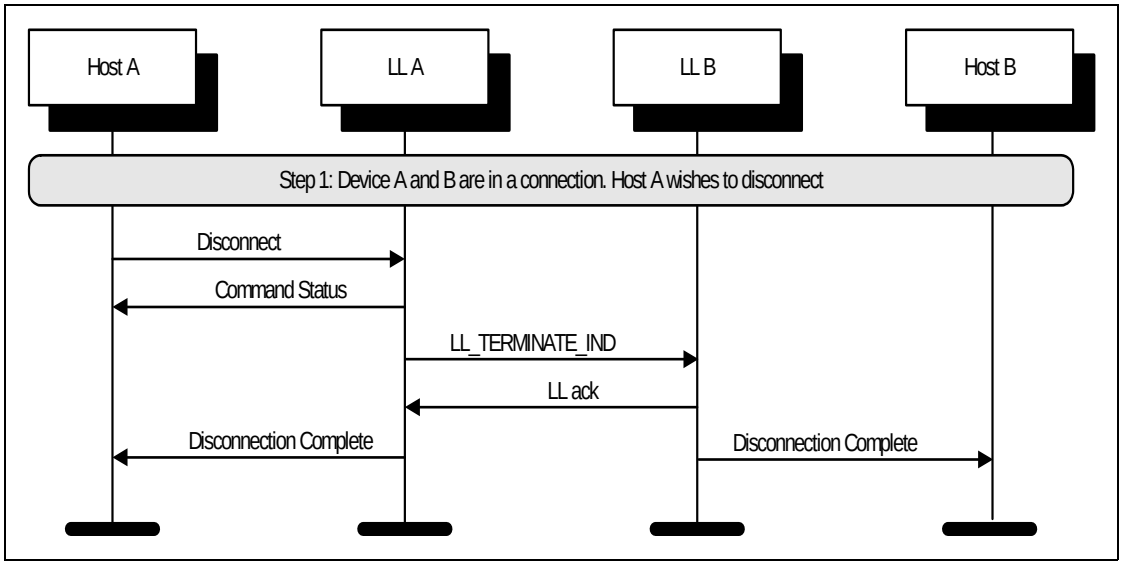



*Figure 6.12: Disconnect from Master*

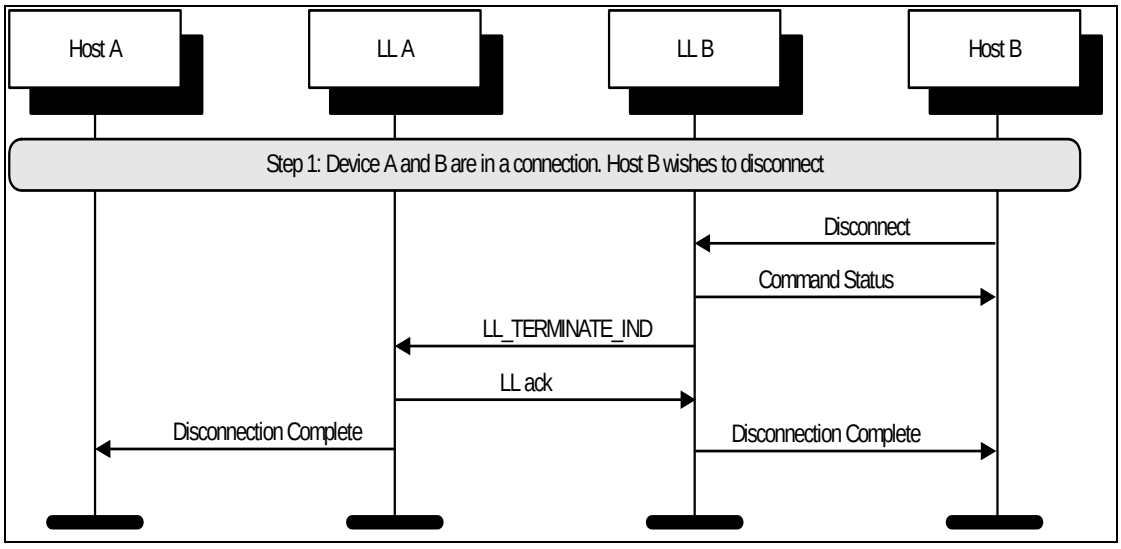



*Figure 6.13: Disconnect from Slave*

# LOW ENERGY LINK LAYER SECURITY

***This part of the specification describes the Link Layer security for Bluetooth low energy.***



Ford Ex. 1052

# CONTENTS

**1 Encryption and Authentication Overview ..... 121**

**2 CCM ..... 122**

2.1 CCM Nonce ..... 122

2.2 Counter Mode Blocks ..... 123

2.3 Encryption Blocks ..... 124

# 1 ENCRYPTION AND AUTHENTICATION OVERVIEW

The Link Layer provides encryption and authentication using Counter with Cipher Block Chaining-Message Authentication Code (CCM) Mode, which shall be implemented consistent with the algorithm as defined in IETF RFC 3610 (http://www.ietf.org/rfc/rfc3610.txt) in conjunction with the AES-128 block cipher as defined in NIST Publication FIPS-197 (http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf). A description of the CCM algorithm can also be found in the NIST Special Publication 800-38C (http://csrc.nist.gov/publications/PubsSPs.html).

This specification uses the same notation and terminology as the IETF RFC except for the Message Authentication Code (MAC) that in this specification is called the Message Integrity Check (MIC) to avoid confusion with the term Media Access Controller.

CCM has two size parameters, M and L. The Link Layer defines these to be:

- M = 4; indicating that the MIC (authentication field) is 4 octets
- L = 2; indicating that the Length field is 2 octets

CCM requires a new temporal key whenever encryption is started. CCM also requires a unique nonce value for each Data Channel PDU protected by a given temporal key. The CCM nonce shall be 13 octets.

The Link Layer connection may be either encrypted and authenticated or unencrypted and unauthenticated. In an encrypted and authenticated connection, all the Data Channel PDUs with a non-zero length Payload shall be encrypted and authenticated. Authentication is performed by appending a MIC field to the Payload. The MIC shall be calculated over the Data Channel PDU's Payload field and the first octet of the header (Part B, Section 2.4) with the NESN, SN and MD bits masked to zero.

Encryption shall be applied to the Data Channel PDU's Payload field and MIC.

Each new Data Channel PDU with a non-zero length Payload shall be decrypted and authenticated before being sent to the Host or processed by the Link Layer. Authentication is unrelated to Link Layer acknowledgement scheme; authentication does not have to be performed before the packet is acknowledged by the Link Layer.

In the unlikely event of an authentication failure being detected, the connection shall be considered lost. The Link Layer shall not send or receive any further packets on that connection. The Link Layer shall exit the Connection State and transition to the Standby State. The Host shall be notified of the loss of connection due to an authentication failure. The peer Link Layer will detect this loss of connection through the supervision timeout procedure.

# 2 CCM

This section provides the details for using the CCM algorithm. As specified, the CCM algorithm requires the payload and some additional parameters to be formatted into the CCM nonce, counter-mode blocks and encryption blocks. The CCM nonce provides uniqueness to each packet. The counter-mode blocks are used to calculate the MIC. The encryption blocks provide the keystream that is used to encrypt the payload and the MIC of the Data Channel PDU.

Sample data of the blocks (see Section 2.2 and Section 2.3) can be found in [Vol 6] Part C, Section  and Section 1.2.

## 2.1 CCM NONCE

The CCM nonce is constructed from a 39-bit *packetCounter*, 1-bit *directionBit* and an 8-octet IV (initialization vector). The format of the 13-octet nonce shall be as shown in Table 2.1.

| Octet | Field | Size (octets) | Value | Description |
|---|---|---|---|---|
| 0 | Nonce0 | 1 | variable | Octet0 (LSO) of *packetCounter* |
| 1 | Nonce1 | 1 | variable | Octet1 of *packetCounter* |
| 2 | Nonce2 | 1 | variable | Octet2 of *packetCounter* |
| 3 | Nonce3 | 1 | variable | Octet3 of *packetCounter* |
| 4 | Nonce4 | 1 | variable | Bit 6 – Bit 0: Octet4 (7 most significant bits of *packetCounter*, with Bit 6 being the most significant bit)<br>Bit7:*directionBit* |
| 5 | Nonce5 | 1 | variable | Octet0 (LSO) of IV |
| 6 | Nonce6 | 1 | variable | Octet1 of IV |
| 7 | Nonce7 | 1 | variable | Octet2 of IV |
| 8 | Nonce8 | 1 | variable | Octet3 of IV |
| 9 | Nonce9 | 1 | variable | Octet4 of IV |
| 10 | Nonce10 | 1 | variable | Octet5 of IV |
| 11 | Nonce11 | 1 | variable | Octet6 of IV |
| 12 | Nonce12 | 1 | variable | Octet7 (MSO) of IV |

*Table 2.1: CCM nonce format*

The Link Layer shall maintain one *packetCounter* per Role for each connection.

For each connection, the *packetCounter* shall be set to zero for the first encrypted Data Channel PDU sent during the encryption start procedure. The

*packetCounter* shall then be incremented by one for each new Data Channel PDU that is encrypted. The *packetCounter* shall not be incremented for retransmissions.

The *directionBit* shall be set to '1' for Data Channel PDUs sent by the master and set to '0' for Data Channel PDUs sent by the slave.

The IV is common for both Roles of a connection. Whenever encryption is started or restarted, a new 8-octet IV shall be used for each pair of communicating devices. The IV is determined as specified in Part B, Section 5.1.3.1.

## 2.2 COUNTER MODE BLOCKS

For calculating the MIC, the multiple counter mode blocks are generated according to the CCM specification. These are referred to as blocks $B_0$ – $B_n$. Table 2.2 defines the format of block $B_0$. Table 2.3 defines the format of block $B_1$ that is devoted to the authentication of the additional authenticated data. Additional B blocks are generated as needed for authentication of the payload.

| Offset (octets) | Field | Size (octets) | Value | Description |
|---|---|---|---|---|
| 0 | Flags | 1 | 0x49 | As per the CCM specification |
| 1 | Nonce | 13 | variable | The nonce as described in Table 2.1.<br>Nonce0 shall have offset 1. Nonce12 shall have offset 13. |
| 14 | Length[MSO] | 1 | 0x00 | The most significant octet of the length of the payload |
| 15 | Length[LSO] | 1 | variable | The least significant octet of the length of the payload |

*Table 2.2: Block $B_0$ format*

| Offset | Field | Size (octets) | Value | Description |
|---|---|---|---|---|
| 0 | AAD_Length[MSO] | 1 | 0x00 | The most significant octet of the length of the additional authenticated data |
| 1 | AAD_Length[LSO] | 1 | 0x01 | The least significant octet of the length of the additional authenticated data |
| 2 | AAD | 1 | variable | The data channel PDU header's first octet with NESN, SN and MD bits masked to 0 |

*Table 2.3: Block $B_1$ format*

| Offset | Field | Size (octets) | Value | Description |
|---|---|---|---|---|
| 3 | Padding | 13 | 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 | These octets are only used to pad the block. They are not part of the packet and never transmitted. |

*Table 2.3: Block $B_1$ format*

## 2.3 ENCRYPTION BLOCKS

The CCM algorithm uses the $A_x$ blocks to generate keystream that is used to encrypt the MIC and the Data Channel PDU payload. Block $A_0$ is always used to encrypt and decrypt the MIC. Block $A_1$ is always used to encrypt and decrypt the first 16 octets of the Payload. Block $A_2$ is always used to encrypt and decrypt the rest of the Payload as needed.

| Offset (octets) | Field | Size (octets) | Value | Description |
|---|---|---|---|---|
| 0 | Flags | 1 | 0x01 | As per the CCM specification |
| 1 | Nonce | 13 | variable | The nonce as described above<br>Nonce0 shall have offset 1. Nonce12 shall have offset 13. |
| 14 | i[MSO] | 1 | variable | The most significant octet of the counter i |
| 15 | i[LSO] | 1 | variable | The least significant octet of the counter i |

*Table 2.4: Block $A_x$ format*

# DIRECT TEST MODE

***This part of the specification describes the Direct Test Mode for RF PHY testing of Bluetooth low energy devices.***


Ford Ex. 1052

# CONTENTS

**1 Introduction ..... 128**

**2 Low Energy Test Scenarios ..... 129**

2.1 Test Sequences ..... 129

2.2 Message Sequence Charts ..... 130

**3 UART Test Interface ..... 132**

3.1 UART Interface Characteristics ..... 132

3.2 UART Functional Description ..... 132

3.3 Commands and Events ..... 133

3.3.1 Command and Event Behavior ..... 133

3.3.2 Commands ..... 133

3.4 Events ..... 134

3.4.1 LE_Test_Status_Event ..... 135

3.4.2 LE_Packet_Report_Event ..... 135

3.5 Timing – Command and Event ..... 136

# 1 INTRODUCTION

Direct Test Mode is used to control the Device Under Test (DUT) and provides a report back to the Tester.

Direct Test Mode shall be set up using one of two alternate methods:

1. over HCI (as defined in Section 2) or
2. through a 2-wire UART interface (as defined in Section 3)

Each DUT shall implement one of the two Direct Test Mode methods in order to test the Low Energy PHY layer. Figure 1.1 illustrates the alternatives for Direct Test Mode setup.

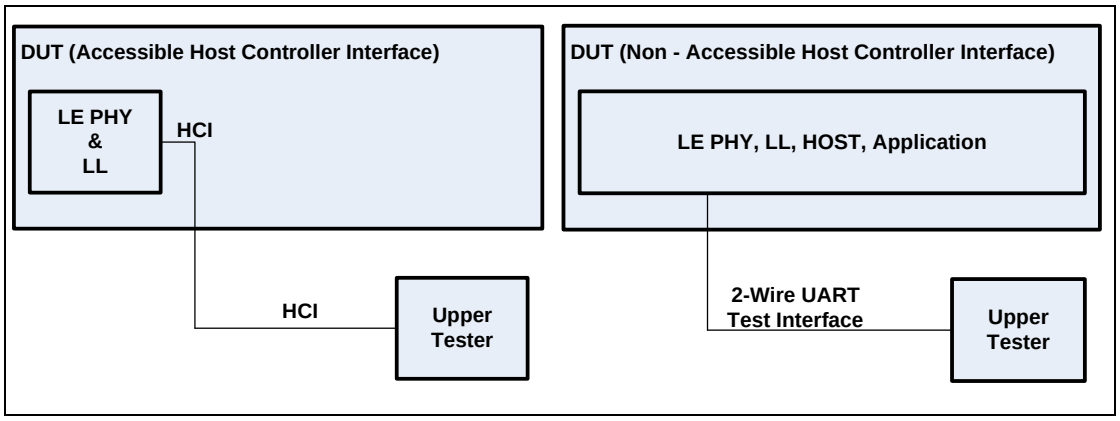



*Figure 1.1: Setup alternatives for LE Direct Test Mode: Designs with accessible HCI (left) and designs without accessible HCI (right)*

Figure 1.2 illustrates the Bluetooth LE Direct Test Mode setup principle using a 2-wire UART interface.

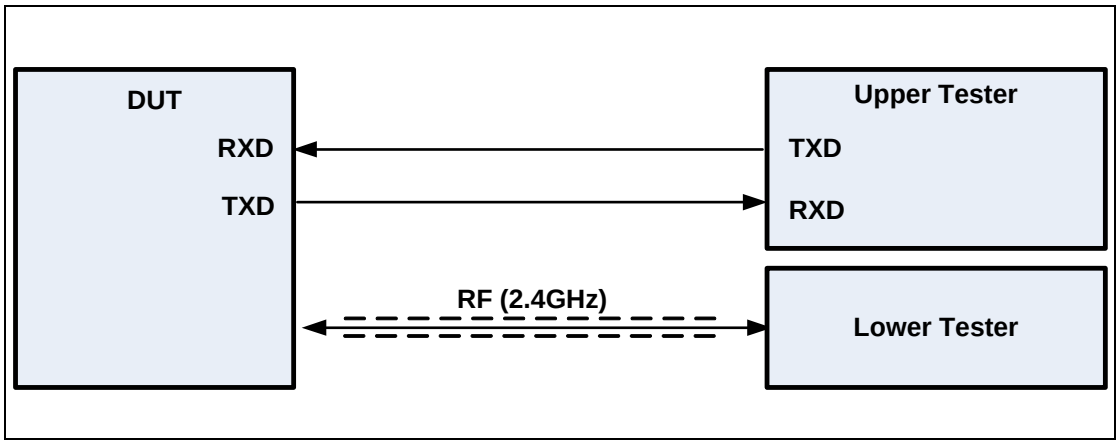



*Figure 1.2: RF PHY test setup for Direct Test Mode (UART control)*

# 2 LOW ENERGY TEST SCENARIOS

## 2.1 TEST SEQUENCES

These sequences are used as routines and used to control an LE DUT with an accessible HCI or a 2-wire UART interface for RF testing.

The following mapping shall be performed from the RF testing commands to HCI commands and events or 2-wire UART commands and events:

| RF Test Command / Event | HCI Command / Event | 2-wire UART Command / Event |
|---|---|---|
| LE_TRANSMITTER_TEST | LE Transmitter Test command | LE Transmitter Test |
| LE_RECEIVER_TEST | LE Receiver Test command | LE Receiver Test |
| LE_TEST_END | LE Test End command | LE Test End |
| LE_STATUS | Command Complete event | LE Test Status |
| LE_PACKET_REPORT | Command Complete event | LE Packet Report |

*Table 2.1: Mapping table of HCI / 2-wire Commands/Events*

The HCI commands and events used in Direct Test Mode are defined in [Vol. 2] Part E, Section 7.8

## 2.2 MESSAGE SEQUENCE CHARTS

### Transmitter Test

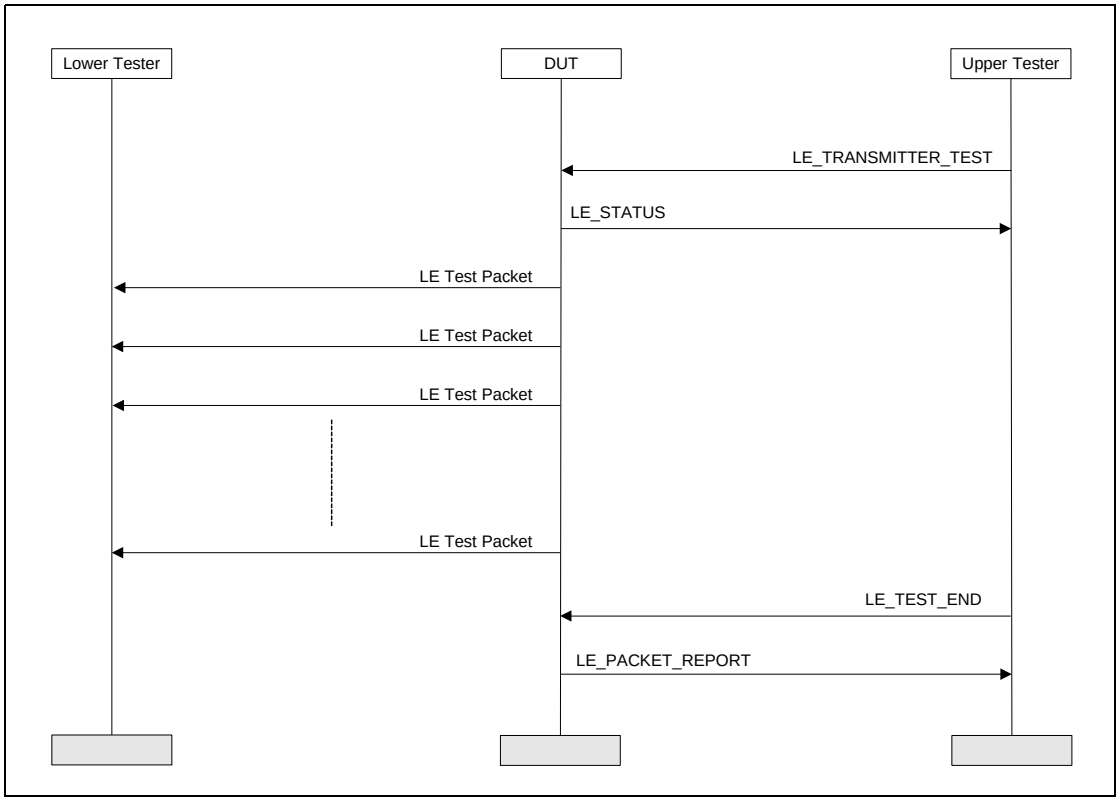



*Figure 2.1: Transmitter Test MSC*

## Receiver Test

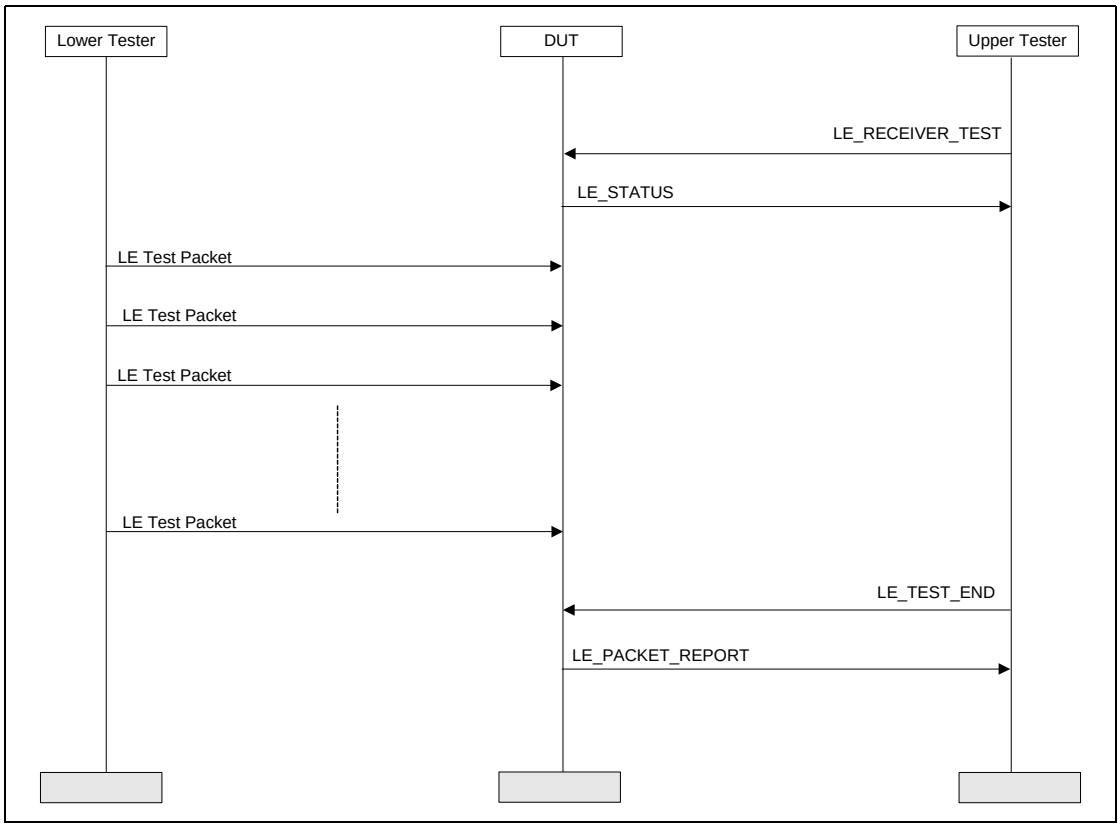



*Figure 2.2: Receiver Test MSC*

# 3 UART TEST INTERFACE

## 3.1 UART INTERFACE CHARACTERISTICS

The UART interface characteristics shall be set to use the following parameters:

- Baud rate: One of the following shall be supported by the DUT:
  1200, 2400, 9600, 14400, 19200, 38400, 57600, 115200
- Number of data bits: 8
- No parity
- 1 stop bit
- No flow control (RTS or CTS)

## 3.2 UART FUNCTIONAL DESCRIPTION

The Upper Tester shall always initiate any a test scenario using the UART interface. The DUT shall respond to the commands from the Upper Tester.

The Upper Tester sends test commands to the DUT. The DUT shall respond with a status event or the packet reporting event.

The Upper Tester shall not transmit further commands before it receives a response from the DUT. If the Upper Tester does not receive a response from the DUT within the time $t_{TIMEOUT}$, the Upper Tester shall transmit a reset command to the DUT and display an appropriate error message.

On reception of a reset command, the DUT shall reset all parameters to their default state.

**Definitions**

- All Commands and Events consist of 16 bits (2 octets).
- The most significant bit is bit number 15.
- The least significant bit is bit number 0.
- The most significant octet is from bit 15 to 8.
- The least significant octet is from bit 7 to 0.
- Commands and Events are sent most significant octet (MSO) first, followed by the least significant octet (LSO).

## 3.3 COMMANDS AND EVENTS

### 3.3.1 Command and Event Behavior

Table 3.1 outlines the set of commands which can be received by the DUT and the corresponding response events that can be transmitted by the DUT.

| Command (DUT RXD) | Event (DUT TXD) |
|---|---|
| LE_Reset | LE_Test_Status SUCCESS<br>LE_Test_Status FAIL |
| LE_Receiver_Test | LE_Test_Status SUCCESS<br>LE_Test_Status FAIL |
| LE_Transmitter_Test | LE_Test_Status SUCCESS<br>LE_Test_Status FAIL |
| LE_Test_End | LE_Packet_Report<br>LE_Test_Status FAIL |

*Table 3.1: 2-Wire command and event behavior*

### 3.3.2 Commands

Command packet format is as shown in Figure 3.1.

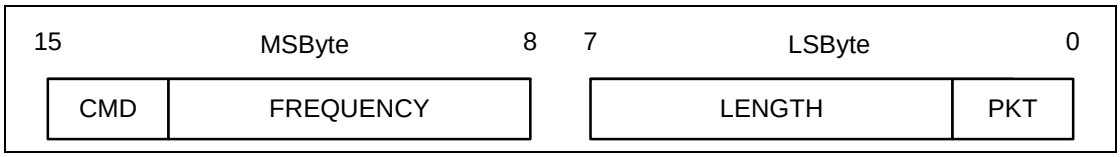



*Figure 3.1: Command message format*

*CMD (command):* *Size: 2 Bits*

| Value $b_1b_0$ | Parameter Description |
|---|---|
| 00 | Reset |
| 01 | Receiver Test |
| 10 | Transmitter Test |
| 11 | Test End |

*Frequency:* *Size: 6 Bits*

| Value | Parameter Description |
|---|---|
| N | When N = 0x00 – 0x27: N = (F-2402)/2 Frequency Range 2402 MHz to 2480 MHz<br>Where N = 0x28 – 0x3F: Reserved |

*Length:* *Size: 6 Bits*

| Value | Parameter Description |
|---|---|
| N | When N = 0x00 – 0x25: Length in octets of payload data in each packet<br>When N = 0x26 – 0x3F: Reserved |

*PKT (Packet Type):* *Size: 2 Bits*

| Value $b_1b_0$ | Parameter Description |
|---|---|
| 00 | PRBS9 Packet Payload |
| 01 | 11110000 Packet Payload |
| 10 | 10101010 Packet Payload |
| 11 | Vendor specific |

## 3.4 EVENTS

There are two types of events sent by the DUT:

1. LE_Test_Status_Event
2. LE_Test_Packet_Report_Event

Event packet format is as shown in Figure 3.2. This packet format is used for both Command Status Events and Packet Report Events.

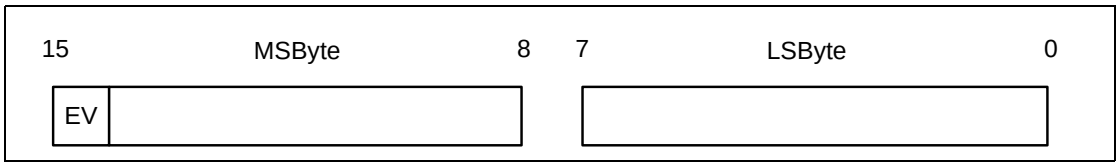



*Figure 3.2: Event packet format*

*EV (Event):* *Size: 1 Bit*

| Value | Parameter Description |
|---|---|
| 0 | LE_Test_Status_Event |
| 1 | LE_Packet_Reporting_Event |

### 3.4.1 LE_Test_Status_Event

The LE_Test_Status_Event packet format is as shown in Figure 3.3.

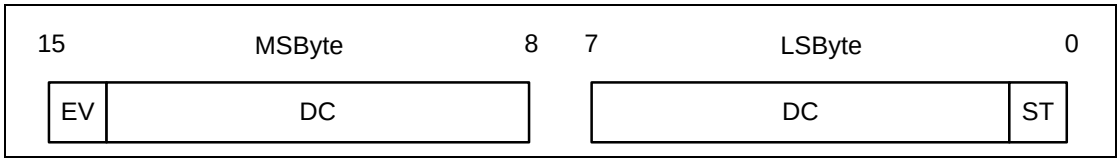



*Figure 3.3: LE Test Status event*

*ST (status):* *Size: 1 Bit*

| Value | Parameter Description |
|---|---|
| 0 | Success |
| 1 | Error |

*DC (don't care):* *Size: 14 Bits*

| Value | Parameter Description |
|---|---|
| xxxxxxxxxxxxx | Value ignored |

### 3.4.2 LE_Packet_Report_Event

The LE_Packet_Report_Event packet format is as shown in Figure 3.4. The *Packet Count* parameter indicates the number of received LE Test Packets. The *Packet Count* in the Packet Report ending a transmitter test shall be 0.

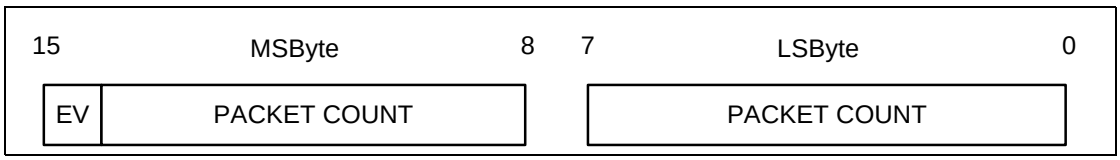



*Figure 3.4: LE Packet Report event*

*PACKET COUNT:* *Size: 15 Bits*

| Value | Parameter Description |
|---|---|
| N | N is the number of packets received<br>Range = 0 to 32767. |

Note**:** The DUT is not responsible for any overflow conditions of the packet counter. That responsibility belongs with the RF PHY Tester or other auxiliary equipment.

## 3.5 TIMING – COMMAND AND EVENT

The timing requirements are as shown in Table 3.2.

| Symbol | Parameter | Min. | Max. | Unit |
|---|---|---|---|---|
| $b_{ERR}$ | Baud rate accuracy | | ±5 | % |
| $t_{MIN}$ | The time between the first and second octet of the command (end of stop bit to start of start bit) | 0 | 5 | ms |
| $t_{RESPONSE}$ | The time from a DUT receiving a command (end of stop bit) until the DUT responds (start of start bit) | 0 | 50 | ms |
| $t_{TURN-AROUND}$ | The time from when the tester receives a response (end of stop bit) until the tester sends another command (start of start bit) | 5 | - | ms |
| $t_{TIMEOUT}$ | The time from when a tester sends a command (end of stop bit) until the tester times out (not having received end of the stop bit in the response) | 51 | 100 | ms |

*Table 3.2: Parameter requirements table for 2-wire UART interface*

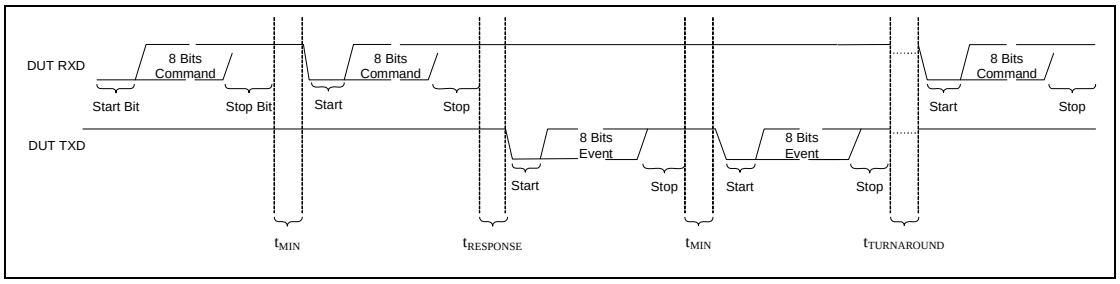



*Figure 3.5: Command and event timing on 2-wire UART interface*

The commands and events shall be transmitted with two 8-bit octets with a maximum time between the 2 transmissions. A timeout is required for no response or an invalid response from the DUT.

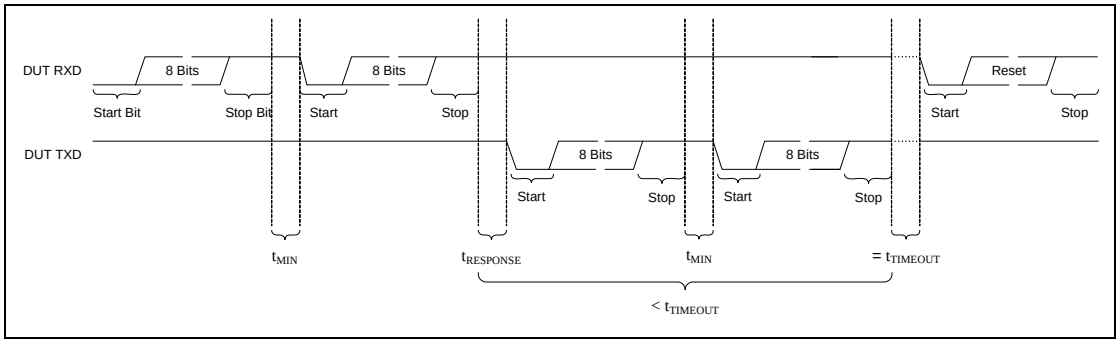



*Figure 3.6: Command and event timing on 2-wire UART interface showing timeout*

