AM_ADDR can be derived in the preceding slot, the slave is still allowed to transmit in the reserved SCO slot.

8.10.8.5.2 Polling in park mode

In the park mode, parked slaves are allowed to send access requests in the access window provided a broadcast packet is received in the preceding master-to-slave slot. Slaves in active mode will not send in the slave-to-master slots following the broadcast packet since they are only allowed to send if addressed specifically.

8.10.8.6 Slot reservation scheme

The SCO link is established by negotiations between the link managers which involves the exchange of important SCO timing parameters like T_{SCO} and D_{SCO} through LMP messages.

8.10.8.7 Broadcast scheme

The master of the piconet can broadcast messages which will reach all slaves. A broadcast packet is characterized by the all-zero AM_ADDR. Each new broadcast message (which may be carried by a number of packets) shall start with the flush indication (L_CH=10).

A broadcast packet is never acknowledged. In an error-prone environment, the master may carry out a number of retransmissions to increase the probability for error-free delivery (see also 8.5.3.5).

In order to support the park mode (as described in 8.10.8.4), a master transmission shall take place at fixed intervals. This master transmission will act as a beacon to which slaves can synchronize. If no traffic takes place at the beacon event, broadcast packets shall be sent. More information is given in 8.10.8.4.

8.10.9 Scatternet

8.10.9.1 General

Multiple piconets may cover the same area. Since each piconet has a different master, the piconets hop independently, each with their own channel hopping sequence and phase as determined by the respective master. In addition, the packets carried on the channels are preceded by different channel access codes as determined by the master device addresses. As more piconets are added, the probability of collisions increases; a graceful degradation of performance results as is common in frequency-hopping spread spectrum systems.

If multiple piconets cover the same area, a unit can participate in two or more overlaying piconets by applying time multiplexing. To participate on the proper channel, it should use the associated master device address and proper clock offset to obtain the correct phase. A Bluetooth unit can act as a slave in several piconets, but only as a master in a single piconet: since two piconets with the same master are synchronized and use the same hopping sequence, they are one and the same piconet. A group of piconets in which connections consists between different piconets is called a scatternet.

A master or slave can become a slave in another piconet by being paged by the master of this other piconet. On the other hand, a unit participating in one piconet can page the master or slave of another piconet. Since the paging unit always starts out as master, a master-slave role exchange is required if a slave role is desired. This is described in the 8.10.9.3.

Copyright © 2002 IEEE. All rights reserved.

LINKPLAY EXHIBIT 1008
Page 122 of 1169

8.10.9.2 Inter-piconet communications

Time multiplexing shall be used to switch between piconets. In case of ACL links only, a unit can request to enter the hold or park mode in the current piconet during which time it may join another piconet by just changing the channel parameters. Units in the sniff mode may have sufficient time to visit another piconet in between the sniff slots. If SCO links are established, other piconets can only be visited in the nonreserved slots in between. This is only possible if there is a single SCO link using HV3 packets. In the four slots in between, one other piconet can be visited. Since the multiple piconets are not synchronized, guard time shall be left to account for misalignment. This means that only 2 slots can effectively be used to visit another piconet in between the HV3 packets.

Since the clocks of two masters of different piconets are not synchronized, a slave unit participating in two piconets has to take care of two offsets that, added to its own native clock, create one or the other master clock. Since the two master clocks drift independently, regular updates of the offsets are required in order for the slave unit to keep synchronization to both masters.

8.10.9.3 Master-slave switch

There are several occasions when a master-slave (MS) switch is desirable. Firstly, a MS switch is necessary when a unit paging the master of an existing piconet wants to join this piconet, since, by definition, the paging unit initially is master of a “small” piconet only involving the pager (master) and the paged (slave) unit. Secondly, when a slave in an existing piconet wants to set up a new piconet, involving itself as master and the current piconet master as slave. The latter case implies a double role of the original piconet master; it becomes a slave in the new piconet while still maintaining the original piconet as master. Thirdly, a much more complicated example is when a slave wants to fully take over an existing piconet, i.e., the switch also involves transfer of other slaves of the existing piconet to the new piconet. Clearly, this can be achieved by letting the new master setup a completely new piconet through the conventional paging scheme. However, that would require individual paging of the old slaves, and, thus, take an unnecessarily long time. Instead, letting the new master utilize timing knowledge of the old master is more efficient. As a consequence of the MS switch, the slaves in the piconet have to be transferred to the new piconet, changing their timing and their hopping scheme.

The MS switch is described below. Prior to the MS switch, encryption if present, shall be stopped in the old piconet. For the third example involving the transfer, new piconet parameters have to be communicated to each slave. The process of this is described below. Unfortunately, even though all the hooks are defined for an efficient transfer at baseband level, there are still many issues that lack sufficient support in the higher layers of the Bluetooth specification (such as how to handle security and transfer all kind of slave information from old to new master). Until all levels of the specification fully supports this kind of transfer, this functionality will have to be taken care of at application layer. These transfer procedures are outside the scope of the baseband specification.

The MS switch procedure will now be described in more detail. For the master and slave involved in the role switch, the MS switch results in a reversal of their TX and RX timing: a TDD switch. Moreover, since the piconet parameters are derived from the device address and clock of the master, an MS switch inherently involves a redefinition of the piconet as well: a piconet switch. The new piconet's parameters are derived from the former slave's device address and clock.

Assume unit A wants to become master; unit B was the former master. Then there are basically two alternative scenarios: either the slave takes the MS switch initiative or the master takes the MS switch initiative. These scenarios are described in 9.3.12.

Both slave A and master B do the TDD switch but keep the former hopping scheme (still using the device address and clock of unit B), so there is no piconet switch yet. The slot offset information sent by slave A is

Copyright © 2002 IEEE. All rights reserved.

not used yet, but is used later in the process. Unit A now becomes the master, unit B the slave. The AM_ADDR formerly used by unit A in its slave role, is now used by slave B.

At the moment of the TDD switch, both units A and B will start a timer with a time out of newconnectionTO. The timer is stopped in slave B as soon as it receives an FHS packet from master A on the TDD-switched channel, the timer is stopped in master A as soon as it receives an ID packet from slave B. If the newconnectionTO expires, the master and slave will return to the old piconet timing and take their old role of master and slave. The FHS packet is sent by master A using the “old” piconet parameters. The AM_ADDR in the FHS packet header is the former AM_ADDR used by unit A. The AM_ADDR carried in the FHS payload is the new AM_ADDR intended for unit B when operating on the new piconet. After the FHS acknowledgment, which consists of the ID packet and is sent by the slave on the old hopping sequence, both master A and slave B turn to the new channel parameters of the new piconet as indicated by the FHS.

Since the old and new masters' clocks are synchronous, the clock information sent in the FHS payload should indicate the new master's clock at the begining of the FHS packet transmission. Furthermore, the 1.25 ms resolution of the clock information given in the FHS packet is not sufficient for aligning the slot boundaries of the two piconets. The slot-offset information in the LMP message previously sent by unit A is used to provide more accurate timing information. The slot offset indicates the delay between the start of the master-to-slave slots of the old and new piconet channels. This timing information ranges from 0 to 1249 µs with a resolution of 1µs. It is used together with the clock information in the FHS packet to accurately position the correlation window when switching to the new master's timing after acknowledgment of the FHS packet. After reception of the FHS packet acknowledgment, the new master A switches to its own timing and sends a POLL packet to verify the switch. Both the master and the slave will start a new timer with a time out of newconnectionTO on FHS packet acknowledgment. The start of this timer shall be aligned with the beginning of the first master TX slot boundary of the new piconet, following the FHS packet acknowledgment. The slave stops the timer when the POLL packet is received; the master stops the timer when the POLL packet is acknowledged. The slave uses a NULL packet to acknowledge the POLL. If no response is received, the master re-sends the POLL packet until newconnectionTO is reached. Should this timer expire, both the slave and the master return to the old piconet timing with the old master and slave roles. The procedure may then start again at the beginning. Aligning the timer with TX boundaries of the new piconet ensures that no unit returns to the old piconet timing in the middle of a master RX slot.

If the new master wishes to take over slaves from the old piconet (which were slaves to the old master B), a piconet switch is enforced on each slave separately. Since the existing slaves already have the correct TDD timing, a TDD switch is not required. Master A sends a slot-offset LMP message to indicate the difference in slot timing of the old and new piconet channel. Thereafter, master A sends an FHS packets and waits for an acknowledgment in the form of an ID packet. When sending the FHS packet, the master starts a timer with a time out of newconnectionTO. The timer is stopped in master A as soon as it receives an ID packet from the slave. Transmission of the FHS packet and the acknowledgment is carried out with the "old" piconet parameters of unit B (compare this to the page hopping scheme used during connection establishment, see 8.10.6.4). Should the timer in master A expire, it may restart the transfer operation. After FHS acknowledgment by the slave, the communication to this slave continues with the new device address and clock of unit A. The FHS packet sent to each slave has the old AM_ADDR in the FHS packet header and their new AM_ADDR in the FHS packet payload (the new AM_ADDR may be identical to the old AM_ADDR).

After reception of the FHS packet acknowledgment, the new master A switches to its own timing and hopping sequence and sends a POLL packet to verify the switch. Both the master and the slave will start a timer with a time out of newconnectionTO on FHS packet acknowledgment. The start of this timer shall be aligned with the beginning of the first master TX slot boundary of the new piconet, following the FHS packet acknowledgment. The slave stops the timer when the POLL packet is received; the master stops the timer when the POLL packet is acknowledged. The slave uses a NULL packet to acknowledge the POLL. If no response is received, the master re sends the POLL packet until newconnectionTO is reached. If the timer expires, both the slave and the master return to the old piconet parameters. The procedure may then be

Copyright © 2002 IEEE. All rights reserved.

LINKPLAY EXHIBIT 1008
Page 124 of 1169

repeated. If an existing slave is out of the range of the new master, master A cannot switch the slave to the new piconet. In that case, the slave loses the connection with the existing piconet after the TDD switch when master B is replaced with master A. As a result, the first message sent by master A being the slot-offset LMP message is never acknowledged by this slave. Due to the link supervision mechanism, the slave will eventually be detached from the old piconet.

Summarized, the MS-switch takes place in two steps: first a TDD switch of the considered master and slave, followed by a piconet switch of the both participants. Then, if so desired, other slaves of the old piconet can be transferred to the new piconet. When a unit have acknowledged the reception of the FHS packet, this unit uses the new piconet parameters defined by the new master and the piconet switch is completed. Note that the SEQN of the first data packet containing a CRC on the new piconet channel is set to 1 (see 8.5.3.2).

A parked slave must be unparked before it can participate in a MS switch. Parked slaves that are transferred to a new piconet shall be activated using the old park parameters, changed to the new piconet parameters, and then returned to the park mode using the new park parameters.

8.10.10 Power management

Features are included into Bluetooth to ensure a low-power operation. These features are both at the microscopic level when handling the packets, and at the macroscopic level using certain operation modes.

8.10.10.1 Packet handling

In order to minimize power consumption, packet handling is minimized both at TX and RX sides. At the TX side, power is minimized by only sending useful data. This means that if only link control information needs to be exchanged, NULL packets will be used. No transmission is carried out at all if there is no link control information or involves a NAK only (NAK is implicit on no reply). If there is data to be sent, the payload length is adapted in order to send only the valid data bytes. At the RX side, packet processing takes place in different steps. If no valid access code is found in the search window, the transceiver returns to sleep. If an access code is found, the receiver unit is woken up and starts to process the packet header. If the HEC fails, the unit will return to sleep after the packet header. A valid header will indicate if a payload will follow and how many slots are involved.

8.10.10.2 Slot occupancy

As was described in 8.4.4, the packet type indicates how many slots a packet may occupy. A slave not addressed in the first slot can go to sleep for the remaining slots the packet may occupy. This can be read from the TYPE code.

8.10.10.3 Low-power modes

In 8.10.8, three modes were described during the CONNECTION state which reduce power consumption. If we list the modes in increasing order of power efficiency then the sniff mode has the higher duty cycle, followed by the hold mode with a lower duty cycle, and finishing with the park mode with the lowest duty cycle.

8.10.11 Link supervision

A connection may break down due to various reasons such as a device moving out of range or a power failure condition. Since this may happen without any prior warning, it is important to monitor the link on both the master and the slave side to avoid possible collisions when the AM_ADDR is reassigned to another slave.

Copyright © 2002 IEEE. All rights reserved.

To be able to supervise link loss, both the master and the slave use link supervision timers, $T_{supervision}$. Upon reception of a packet that passes the HEC check and has the correct AM_ADDR, the timer is reset. If at any time in connection state, the timer reaches the *supervisionTO* value, the connection is reset. The same timeout value is used for both SCO and ACL connections.

The timeout period, *supervisionTO*, is negotiated at the LM level. Its value is chosen so that the supervision timeout will be longer than hold and sniff periods. Link supervision of a parked slave will be done by unparking and re-parking the slave.

8.11 Hop selection

In total, ten types of hopping sequences are defined: five for the 79-hop and five for the 23-hop system, respectively. Using the notation of parentheses for figures related to the 23-hop system, these sequences are as follows:

— A page hopping sequence with 32 (16) unique wake-up frequencies distributed equally over the 79 (23) MHz, with a period length of 32 (16);

— A page response sequence covering 32 (16) unique response frequencies that all are in an one-to-one correspondence to the current page hopping sequence. The master and slave use different rules to obtain the same sequence;

— An inquiry sequence with 32 (16) unique wake-up frequencies distributed equally over the 79 (23) MHz, with a period length of 32 (16);

— A inquiry response sequence covering 32 (16) unique response frequencies that all are in an one-to-one correspondence to the current inquiry hopping sequence.

— A channel hopping sequence which has a very long period length, which does not show repetitive patterns over a short time interval, but which distributes the hop frequencies equally over the 79 (23) MHz during a short time interval.

For the page hopping sequence, it is important that we can easily shift the phase forward or backward, so we need a 1-1 mapping from a counter to the hop frequencies. For each case, both a hop sequence from master to slave and from slave to master are required.

The inquiry and inquiry response sequences always utilizes the GIAC LAP as lower address part and the DCI (see 8.5.4) as upper address part in deriving the hopping sequence, even if it concerns a DIAC inquiry.

8.11.1 General selection scheme

The selection scheme consists of two parts:

— Selecting a sequence

— Mapping this sequence on the hop frequencies

The general block diagram of the hop selection scheme is shown in Figure 59. The mapping from the input to a particular hop frequency is performed in the selection box. Basically, the input is the native clock and the current address. In CONNECTION state, the native clock (CLKN) is modified by an offset to equal the master clock (CLK). Only the 27 MSBs of the clock are used. In the page and inquiry substates, all 28 bits of the clock are used. However, in page substate the native clock will be modified to the master's estimate of the paged unit.

Copyright © 2002 IEEE. All rights reserved.

LINKPLAY EXHIBIT 1008
Page 126 of 1169

The address input consists of 28 bits, i.e., the entire LAP and the four LSBs of the UAP. In CONNECTION state, the address of the master is used. In page substate the address of the paged unit is used. When in inquiry substate, the UAP/LAP corresponding to the GIAC is used. The output constitutes a pseudo-random sequence, either covering 79 hop or 23 hops, depending on the state.

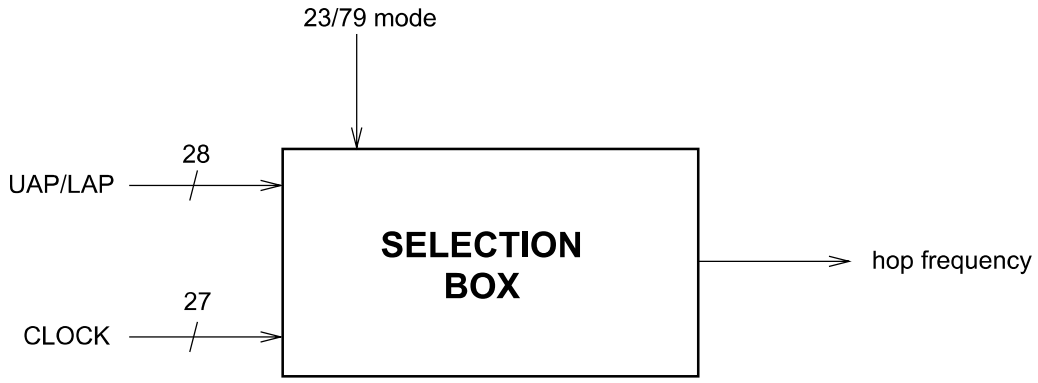


Figure 59—General block diagram of hop selection scheme

For the 79-hop system, the selection scheme chooses a segment of 32 hop frequencies spanning about 64 MHz and visits these hops once in a random order. Next, a different 32-hop segment is chosen, etc. In case of the page, page scan, or page response substates, the same 32-hop segment is used all the time (the segment is selected by the address; different units will have different paging segments). In connection state, the output constitutes a pseudo-random sequence that slides through the 79 hops or 23 hops, depending on the selected hop system. For the 23-hop systems, the segment size is 16. The principle is depicted in Figure 60.

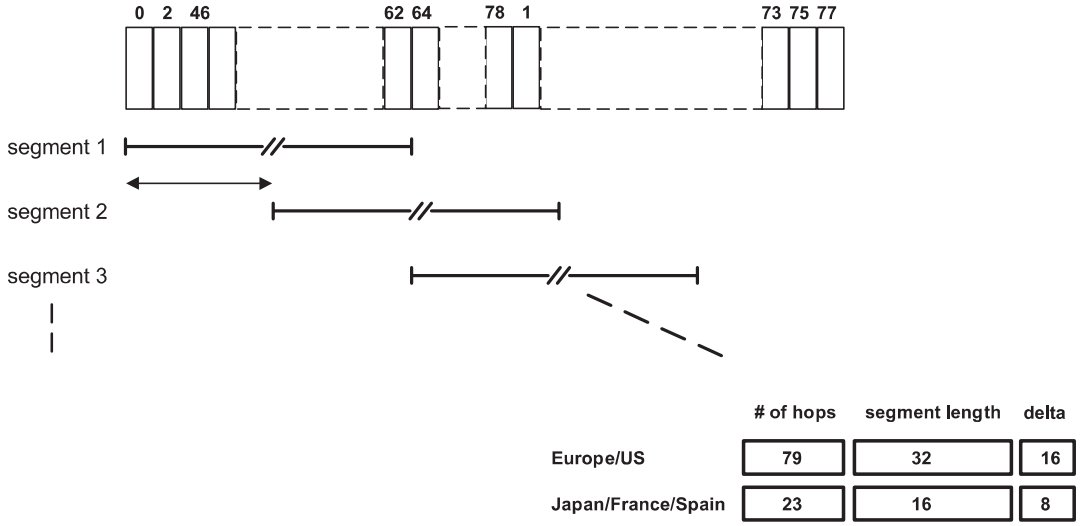


Figure 60—Hop selection scheme in CONNECTION state

Copyright © 2002 IEEE. All rights reserved.

8.11.2 Selection kernel

The hop selection kernels for the 79-hop system and the 23-hop system are shown in Figure 61 and Figure 62, respectively. The *X* input determines the phase in the 32-hop segment, whereas *Y*1 and *Y*2 selects between master-to-slave and slave-to-master transmission. The inputs *A* to *D* determine the ordering within the segment, the inputs *E* and *F* determine the mapping onto the hop frequencies. The kernel addresses a register containing the hop frequencies. This list should be created such that first all even hop frequencies are listed and then all odd hop frequencies. In this way, a 32-hop segment spans about 64 MHz, whereas a 16-hop segment spans the entire 23 MHz.

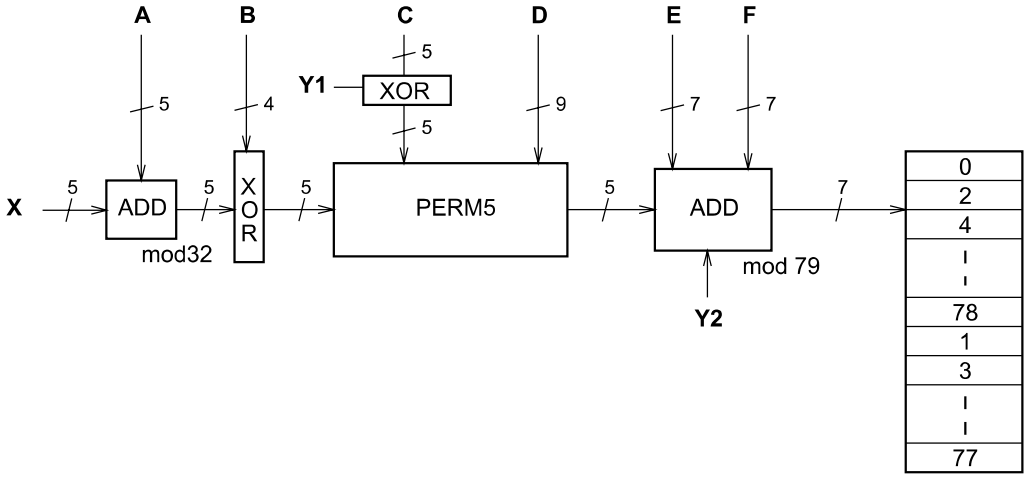


Figure 61—Block diagram of hop selection kernel for the 79-hop system

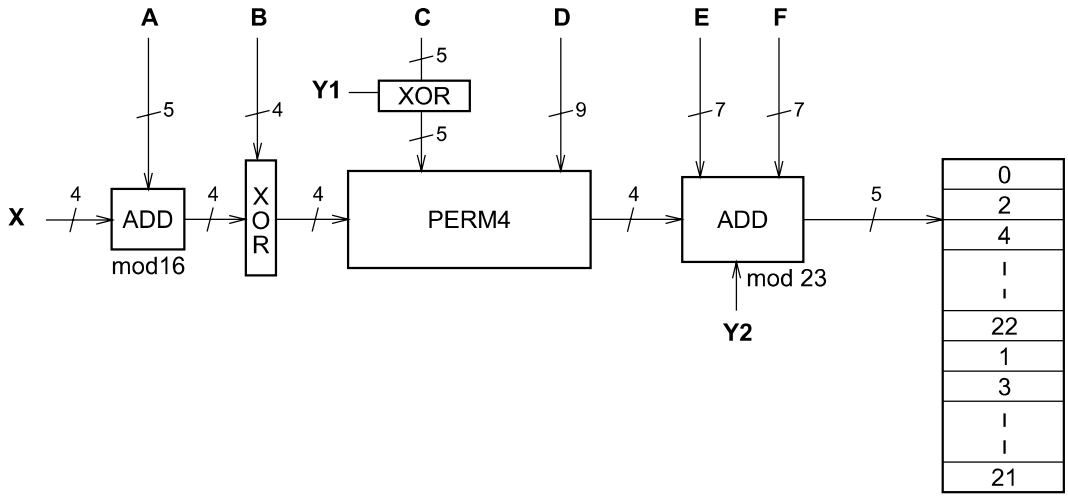


Figure 62—Block diagram of hop selection kernel for the 23-hop system

Copyright © 2002 IEEE. All rights reserved.

LINKPLAY EXHIBIT 1008
Page 128 of 1169

The selection procedure consists of an addition, an XOR operation, a permutation operation, an addition, and finally a register selection. In the remainder of this chapter, the notation A_i is used for bit i of the BD_ADDR.

8.11.2.1 First addition operation

The first addition operation only adds a constant to the phase and applies a modulo 32 or a modulo 16 operation. For the page hopping sequence, the first addition is redundant since it only changes the phase within the segment. However, when different segments are concatenated (as in the channel hopping sequence), the first addition operation will have an impact on the resulting sequence.

8.11.2.2 XOR operation

Let *Z*' denote the output of the first addition. In the XOR operation, the four LSBs of *Z*' are modulo-2 added to the address bits $A_{22–19}$. The operation is illustrated in Figure 63.

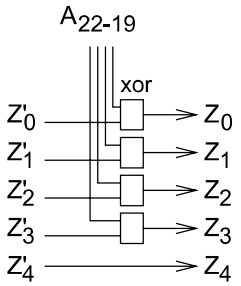


Figure 63—XOR operation for the 79-hop system.
(The 23-hop system is the same except for the Z'_4/Z_4 wire that does not exist.)

8.11.2.3 Permutation operation

The permutation operation involves the switching from 5 inputs to 5 outputs for the 79 hop system and from 4 inputs to 4 outputs for 23 hop system, in a manner controlled by the control word. In Figure 64 and Figure 65 the permutation or switching box is shown. It consists of 7 stages of butterfly operations. Table 29 and Table 30 shows the control of the butterflies by the control signals *P*. Note that $P_{0–8}$ corresponds to $D_{0–8}$, and, P_{i+9} corresponds to $C_i \oplus Y1$ for $i = 0 \ldots 4$ in Figure 61 and Figure 62.

The *Z* input is the output of the XOR operation as described in 8.11.2.2. The butterfly operation can be implemented with multiplexers as depicted in Figure 66.

Copyright © 2002 IEEE. All rights reserved.

Table 29—Control of the butterflies for the 79-hop system

Control signal	Butterfly		Control signal	Butterfly
P_0	$\{Z_0,Z_1\}$		P_8	$\{Z_1,Z_4\}$
P_1	$\{Z_2,Z_3\}$		P_9	$\{Z_0,Z_3\}$
P_2	$\{Z_1,Z_2\}$		P_{10}	$\{Z_2,Z_4\}$
P_3	$\{Z_3,Z_4\}$		P_{11}	$\{Z_1,Z_3\}$
P_4	$\{Z_0,Z_4\}$		P_{12}	$\{Z_0,Z_3\}$
P_5	$\{Z_1,Z_3\}$		P_{13}	$\{Z_1,Z_2\}$
P_6	$\{Z_0,Z_2\}$			
P_7	$\{Z_3,Z_4\}$		—	—

Table 30—Control of the butterflies for the 23-hop system

Control signal	Butterfly		Control signal	Butterfly
P_0	$\{Z_0,Z_1\}$		P_8	$\{Z_0,Z_2\}$
P_1	$\{Z_2,Z_3\}$		P_9	$\{Z_1,Z_3\}$
P_2	$\{Z_0,Z_3\}$		P_{10}	$\{Z_0,Z_3\}$
P_3	$\{Z_1,Z_2\}$		P_{11}	$\{Z_1,Z_2\}$
P_4	$\{Z_0,Z_2\}$		P_{12}	$\{Z_0,Z_1\}$
P_5	$\{Z_1,Z_3\}$		P_{13}	$\{Z_2,Z_3\}$
P_6	$\{Z_0,Z_1\}$		—	—
P_7	$\{Z_2,Z_3\}$		—	—

Copyright © 2002 IEEE. All rights reserved.

LINKPLAY EXHIBIT 1008
Page 130 of 1169

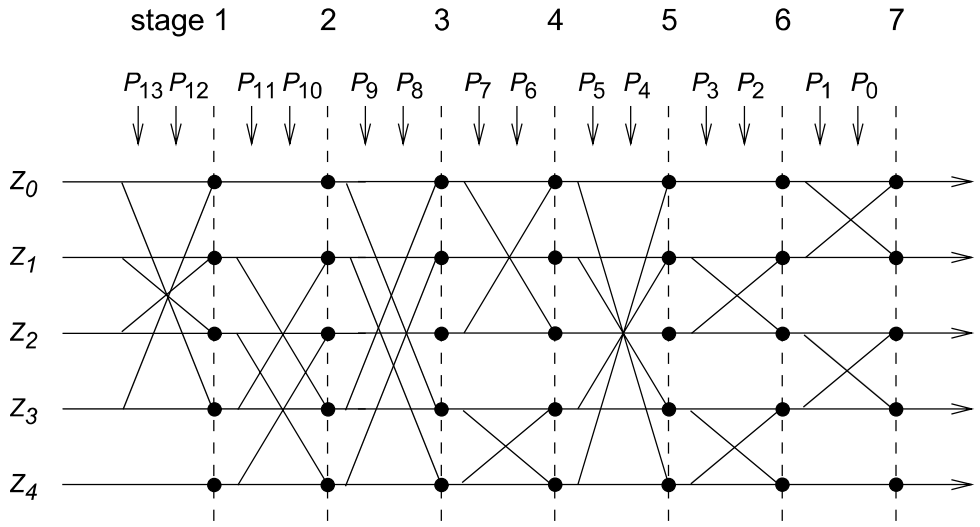


Figure 64—Permutation operation for the 79-hop system

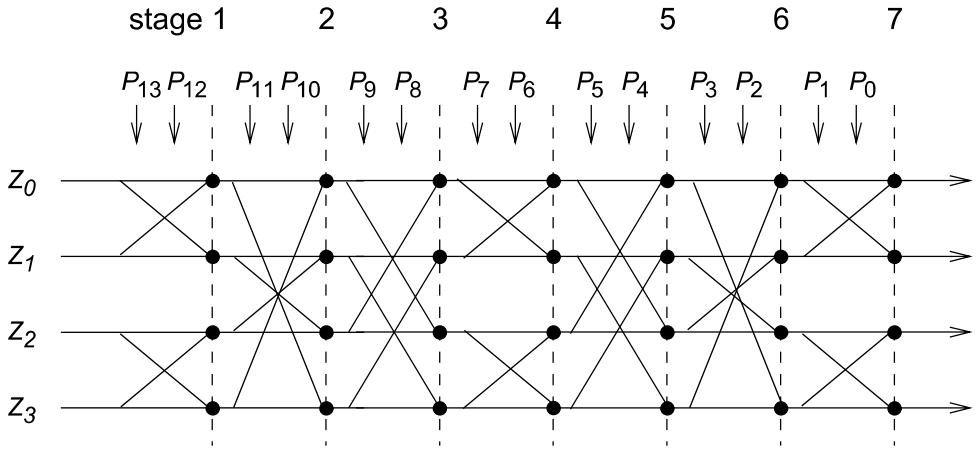


Figure 65—Permutation operation for the 23-hop system

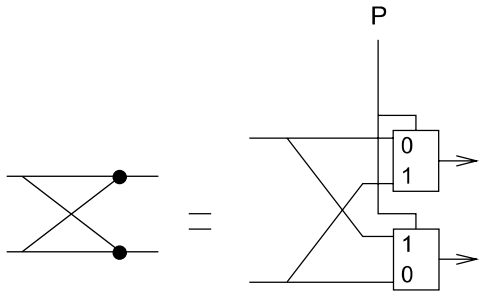


Figure 66—Butterfly implementation

Copyright © 2002 IEEE. All rights reserved.
LINKPLAY EXHIBIT 1008
Page 131 of 1169

8.11.2.4 Second addition operation

The addition operation only adds a constant to the output of the permutation operation. As a result, the 16-hop or 32-hop segment is mapped differently on the hop frequencies. The addition is applied modulo 79 or modulo 23 depending on the system type.

8.11.2.5 Register bank

The output of the adder addresses a bank of 79 or 23 registers. The registers are loaded with the synthesizer code words corresponding to the hop frequencies 0 to 78 or 0 to 22. Note that the upper half of the bank contains the even hop frequencies, whereas the lower half of the bank contains the odd hop frequencies.

8.11.3 Control word

In the following section X_{j-i}, $i<j$, will denote bits i, $i+1$,...,j of the bit vector X. By convention, X_0 is the least significant bit of the vector X.

The control word P of the kernel is controlled by the overall control signals X, $Y1$, $Y2$, and A to F as illustrated in Figure 61 and Figure 62. During paging and inquiry, the inputs A to E use the address values as given in the corresponding columns of Table 31 and Table 32. In addition, the inputs X, $Y1$ and $Y2$ are used. The F input is unused. In the 79-hop system, the clock bits CLK_{6-2} (i.e., input X) specifies the phase within the length 32 sequence, while for the 23-hop system, CLK_{5-2} specifies the phase within the length 16 sequence. For both systems, CLK_1 (i.e., inputs $Y1$ and $Y2$) is used to select between TX and RX. The address inputs determine the sequence order within segments. The final mapping onto the hop frequencies is determined by the register contents.

In the following we will distinguish between three types of clocks: the piconet's master clock, the Bluetooth unit's native clock, and the clock estimate of a paged Bluetooth unit. These types are marked in the following way:

1) CLK_{27-0}: Master clock of the current piconet.
2) $CLKN_{27-0}$: Native clock of the unit.
3) $CLKE_{27-0}$: The paging unit's estimate of the paged unit's native clock.

During the CONNECTION state, the inputs A, C, and D result from the address bits being bit-wise XORed with the clock bits as shown in the "Connection state" column of Table 31 and Table 32 (the two MSBs are XORed together, the two second MSBs are XORed together, etc.). Consequently, after every 32 (16) time slots, a new length 32 (16) segment is selected in the 79-hop (23-hop) system. The sequence order within a specific segment will not be repeated for a very long period. Thus, the overall hopping sequence consists of concatenated segments of 32-hops each. Since each 32-hop sequence spans more than 80% of the 79 MHz band, the desired frequency spreading over a short time interval is obtained.

Copyright © 2002 IEEE. All rights reserved.

LINKPLAY EXHIBIT 1008
Page 132 of 1169

Table 31—Control for 79-hop system

	Page scan/ inquiry scan	Page/inquiry	Page response (master/slave) and Inquiry response	Connection state
X	CLKN_{16-12} / $Xir^{(79)}_{4-0}$	$Xp^{(79)}_{4-0}/Xi^{(79)}_{4-0}$	$Xprm^{(79)}_{4-0}/Xprs^{(79)}_{4-0}/Xir^{(79)}_{4-0}$	CLK_{6-2}
*Y*1	0	$\mathrm{CLKE}_1/\mathrm{CLKN}_1$	$\mathrm{CLKE}_1/\mathrm{CLKN}_1/1$	CLK_1
*Y*2	0	$32\times\mathrm{CLKE}_1$/ $32\times\mathrm{CLKN}_1$	$32\times\mathrm{CLKE}_1$ / $32\times\mathrm{CLKE}_1$ 32×1	$32\times\mathrm{CLK}_1$
A	A_{27-23}	A_{27-23}	A_{27-23}	$A_{27-23}\oplus\mathrm{CLK}_{25-21}$
B	A_{22-19}	A_{22-19}	A_{22-19}	A_{22-19}
C	$A_{8,6,4,2,0}$	$A_{8,6,4,2,0}$	$A_{8,6,4,2,0}$	$A_{8,6,4,2,0}\oplus\mathrm{CLK}_{20-16}$
D	A_{18-10}	A_{18-10}	A_{18-10}	$A_{18-10}\oplus\mathrm{CLK}_{15-7}$
E	$A_{13,11,9,7,5,3,1}$	$A_{13,11,9,7,5,3,1}$	$A_{13,11,9,7,5,3,1}$	$A_{13,11,9,7,5,3,1}$
F	0	0	0	$16\times\mathrm{CLK}_{27-7}$ mod 79

Table 32—Control for 23-hop system

	Page scan/ inquiry scan	Page/inquiry	Page response (master/slave) and inquiry response	Connection state
X	CLKN_{15-12} / $Xir^{(23)}_{3-0}$	$Xp^{(23)}_{3-0}/Xi^{(23)}_{3-0}$	$Xprm^{(23)}_{3-0}/Xprs^{(23)}_{3-0}/Xir^{(23)}_{3-0}$	CLK_{5-2}
*Y*1	0	$\mathrm{CLKE}_1/\mathrm{CLKN}_1$	$\mathrm{CLKE}_1/\mathrm{CLKN}_1/1$	CLK_1
*Y*2	0	$16\times\mathrm{CLKE}_1$/ $16\times\mathrm{CLKN}_1$	$16\times\mathrm{CLKE}_1$ / $16\times\mathrm{CLKE}_1$ 16×1	$16\times\mathrm{CLK}_1$
A	A_{27-23}	A_{27-23}	A_{27-23}	$A_{27-23}\oplus\mathrm{CLK}_{25-21}$
B	A_{22-19}	A_{22-19}	A_{22-19}	A_{22-19}
C	$A_{8,6,4,2,0}$	$A_{8,6,4,2,0}$	$A_{8,6,4,2,0}$	$A_{8,6,4,2,0}\oplus\mathrm{CLK}_{20-16}$
D	A_{18-10}	A_{18-10}	A_{18-10}	$A_{18-10}\oplus\mathrm{CLK}_{15-7}$
E	$A_{13,11,9,7,5,3,1}$	$A_{13,11,9,7,5,3,1}$	$A_{13,11,9,7,5,3,1}$	$A_{13,11,9,7,5,3,1}$
F	0	0	0	$6\times\mathrm{CLK}_{27-6}$ mod 23

8.11.3.1 Page scan and Inquiry scan substates

In page scan, the Bluetooth device address of the scanning unit is used as address input. In inquiry scan, the GIAC LAP and the four LSBs of the DCI (as A_{27-24}), are used as address input for the hopping sequence. Naturally, for the transmitted access code and in the receiver correlator, the appropriate GIAC or DIAC is used. The application decides which inquiry access code to use depending on the purpose of the inquiry.

Copyright © 2002 IEEE. All rights reserved.

The five X input bits vary depending on the current state of the unit. In the page scan and inquiry scan substates, the native clock (CLKN) is used. In CONNECTION state the master clock (CLK) is used as input. The situation is somewhat more complicated for the other states.

8.11.3.2 Page substate

In the page substate of the 79-hop system, the paging unit shall start using the **A-**train, i.e., $\{f(k-8), \ldots, f(k), \ldots, f(k+7)\}$, where $f(k)$ is the source's estimate of the current receiver frequency in the paged unit. Clearly, the index k is a function of all the inputs in Figure 61. There are 32 possible paging frequencies within each 1.28 second interval. Half of these frequencies belongs to the **A-**train, the rest (i.e., $\{f(k+8), \ldots, f(k+15), f(k-16), \ldots, f(k-9)\}$) belongs to the **B-**train. In order to achieve the **-**8 offset of the **A-**train, a constant of 24 can be added to the clock bits (which is equivalent to **-**8 due to the modulo 32 operation). Clearly, the **B-**train may be accomplished by setting the offset to 8. A cyclic shift of the order within the trains is also necessary in order to avoid a possible repetitive mismatch between the paging and scanning units. Thus,

$$Xp^{(79)} = [\mathrm{CLKE}_{16-12} + k_{offset} + (\mathrm{CLKE}_{4-2,0} - \mathrm{CLKE}_{16-12}) \bmod 16] \bmod 32, \quad (2)$$

where

$$k_{offset} = \begin{cases} 24 & \text{A-train,} \\ 8 & \text{B-train.} \end{cases} \quad (3)$$

Alternatively, each switch between the **A-** and **B-**trains may be accomplished by adding 16 to the current value of k_{offset} (originally initialized with 24).

In the page substate of the 23-hop system, the paging unit makes use of the **A-**train only. A constant offset of 8 is used in order to start with $f(k-8)$. Moreover, only four bits are needed since the additions are modulo 16. Consequently,

$$Xp^{(23)} = [\mathrm{CLKE}_{15-12} + 8 + \mathrm{CLKE}_{4-2,0}] \bmod 16, \quad (4)$$

8.11.3.3 Page response

8.11.3.3.1 Slave response

A unit in the page scan substate recognizing its own access code enters the slave response substate. In order to eliminate the possibility of loosing the link due to discrepancies of the native clock CLKN and the master's clock estimate CLKE, the four bits CLKN_{16-12} shall be frozen at their current value. The value is frozen to the content it has in the slot where the recipient's access code is detected. Note that the actual native clock is *not* stopped; it is merely the values of the bits used for creating the X-input that are kept fixed for a while. In the sequel, a frozen value is marked by an asterisk (*).

For each response slot the paged unit will use an X-input value one larger (modulo 32 or 16) than in the preceding response slot. However, the first response is made with the X-input kept at the same value as it was when the access code was recognized. Let N be a counter starting at zero. Then, the X-input in the $(N+1)$-th response slot (the first response slot being the one immediately following the page slot now responding to) of the slave response substate becomes

$$Xprs^{(79)} = [\mathrm{CLKN}^*_{16-12} + N] \bmod 32, \quad (5)$$

Copyright © 2002 IEEE. All rights reserved.

LINKPLAY EXHIBIT 1008
Page 134 of 1169

and

$$Xprs^{(23)} = [\mathrm{CLKN}^*_{15-12} + N] \mod 16, \tag{6}$$

for the 79-hop and 23-hop systems, respectively. The counter N is set to zero in the slot where the slave acknowledges the page (see Figure 52 and Figure 53). Then, the value of N is increased by one each time CLKN_1 is set to zero, which corresponds to the start of a master TX slot. The *X*-input is constructed this way until the first accepted FHS packet is received *and* the immediately following response packet has been transmitted. After this the slave enters the CONNECTION state using the parameters received in the FHS packet.

8.11.3.3.2 Master response

The paging unit enters master response substate upon receiving a slave response. Clearly, also the master must freeze its estimated slave clock to the value that triggered a response from the paged unit. It is equivalent to using the values of the clock estimate when receiving the slave response (since only CLKE_1 will differ from the corresponding page transmission). Thus, the values are frozen when the slave ID packet is received. In addition to the used clock bits, also the current value of k_{offset} shall be frozen. The master will adjust its *X*-input in the same way the paged unit does, i.e., by incrementing this value by one for each time CLKE_1 is set to zero. The first increment shall be done before sending the FHS packet to the paged unit. Let N be a counter starting at one. The rules for forming the *X*-inputs become

$$\begin{aligned} Xprm^{(79)} = [&\mathrm{CLKE}^*_{16-12} + k_{offset}{}^* + \\ &(\mathrm{CLKE}^*_{4-2,0} - \mathrm{CLKE}^*_{16-12}) \bmod 16 + N] \mod 32, \end{aligned} \tag{7}$$

and

$$Xprm^{(23)} = [\mathrm{CLKE}^*_{15-12} + 8 + \mathrm{CLKE}^*_{4-2,0} + N] \mod 16, \tag{8}$$

for the 79-hop and 23-hop systems, respectively. The value of N is increased each time CLKE_1 is set to zero, which corresponds to the start of a master TX slot.

8.11.3.4 Inquiry substate

The *X*-input of the inquiry substate is quite similar to what is used in the page substate. Since no particular unit is addressed, the native clock CLKN of the inquirer is used. Moreover, which of the two train offsets to start with is of no real concern in this state. Consequently,

$$Xi^{(79)} = [\mathrm{CLKN}_{16-12} + k_{offset} + (\mathrm{CLKN}_{4-2,0} - \mathrm{CLKN}_{16-12}) \mod 16] \mod 32, \tag{9}$$

where k_{offset} is defined by Equation (3). The initial choice of the offset is arbitrary. For the 23-hop system,

$$Xi^{(23)} = [\mathrm{CLKN}_{15-12} + 8 + \mathrm{CLKN}_{4-2,0}] \mod 16, \tag{10}$$

The GIAC LAP and the four LSBs of the DCI (as A_{27-24}) are used as address input for the hopping sequence generator.

Copyright © 2002 IEEE. All rights reserved.
LINKPLAY EXHIBIT 1008
Page 135 of 1169

8.11.3.5 Inquiry response

The inquiry response substate is similar to the slave response substate with respect to the X-input. However, there is no need to freeze the clock input, thus

$$Xir^{(79)} = [\mathrm{CLKN}_{16-12} + N] \mod 32, \tag{11}$$

and

$$Xir^{(23)} = [\mathrm{CLKN}_{15-12} + N] \mod 16, \tag{12}$$

for the 79-hop and 23-hop systems, respectively. Furthermore, the counter N is increased not on CLKN_1 basis, but rather after each FHS packet has been transmitted in response to the inquiry. There is no restriction on the initial value of N as it is independent of the corresponding value in the inquiring unit.

The GIAC LAP and the four LSBs of the DCI (as A_{27-24}) are used as address input for the hopping sequence generator. The other input bits to the generator are the same as in the case of page response.

8.11.3.6 Connection state

In CONNECTION state, the clock bits to use in the channel hopping sequence generation are always according to the master clock, CLK. The address bits are taken from the Bluetooth device address of the master.

8.12 Bluetooth audio

On the Bluetooth air-interface, either a 64 kb/s log PCM format (A-law or μ-law) is used, or a 64 kb/s CVSD (Continuous Variable Slope Delta Modulation) is used. The latter format applies an adaptive delta modulation algorithm with syllabic companding.

The voice coding on the line interface should have a quality equal to or better than the quality of 64 kb/s log PCM.

Table 33 summarizes the voice coding schemes supported on the air interface. The appropriate voice coding scheme is selected after negotiations between the Link Managers.

Table 33—Voice coding schemes supported on the air interface

Voice Codecs	
linear	CVSD
8-bit logarithmic	A-law
	μ-law

8.12.1 LOG PCM CODEC

Since the voice channels on the air-interface can support a 64 kb/s information stream, a 64 kb/s log PCM traffic can be used for transmission. Either A-law or μ-law compression can be applied. In the event that the

Copyright © 2002 IEEE. All rights reserved.

LINKPLAY EXHIBIT 1008
Page 136 of 1169

line interface uses A-law and the air interface uses μ-law or vice versa, a conversion from A-law to μ-law is performed. The compression method follows ITU-T Recommendation G.711 (11/88).

8.12.2 CVSD CODEC

A more robust format for voice over the air interface is a delta modulation. This modulation scheme follows the waveform where the output bits indicate whether the prediction value is smaller or larger then the input waveform. To reduce slope overload effects, syllabic companding is applied: the step size is adapted according to the average signal slope. The input to the CVSD encoder is 64 ksamples/s linear PCM. Block diagrams of the CVSD encoder and CVSD decoder are shown in Figure 67, Figure 68 and Figure 69. The system is clocked at 64 kHz.

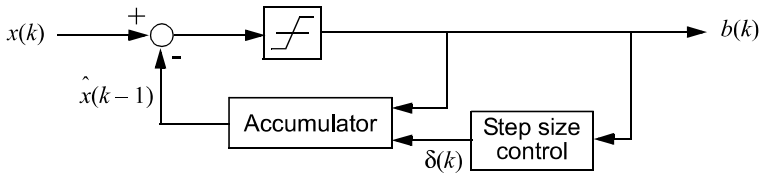


Figure 67—Block diagram of CVSD encoder with syllabic companding

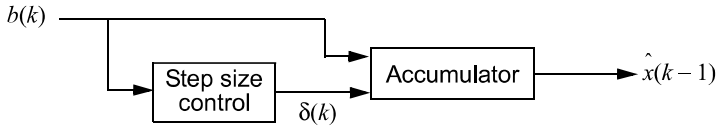


Figure 68—Block diagram of CVSD decoder with syllabic companding

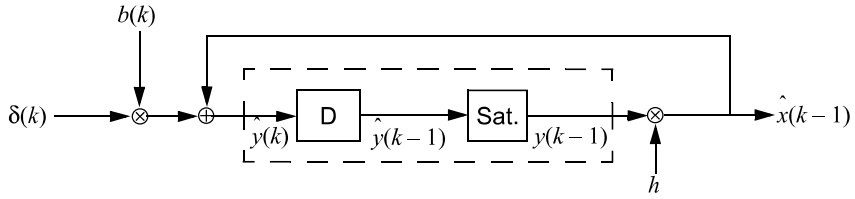


Figure 69—Accumulator procedure

Let $\mathrm{sgn}(x) = 1$ for $x \geq 0$, otherwise, $\mathrm{sgn}(x) = -1$. On air these numbers are represented by the sign bit; i.e. negative numbers are mapped on "1" and positive numbers are mapped on "0". Denote the CVSD encoder output bit $b(k)$, the accumulator contents $y(k)$, and the step size $\delta(k)$. Furthermore, let h be the decay factor for the accumulator, let β denote the decay factor for the step size, and, let α be the syllabic companding parameter. The latter parameter monitors the slope by considering the K most recent output bits.

Copyright © 2002 IEEE. All rights reserved.

Let

$$\hat{x}(k) = hy(k). \tag{13}$$

Then, the CVSD encoder internal state is updated according to the following set of equations:

$$b(k) = \text{sgn}\{x(k) - \hat{x}(k-1)\}, \tag{14}$$

$$\alpha = \begin{cases} 1, & \text{if } J \text{ bits in the last } K \text{ output bits are equal,} \\ 0, & \text{otherwise,} \end{cases} \tag{15}$$

$$\delta(k) = \begin{cases} \min\{\delta(k-1) + \delta_{min}, \delta_{max}\}, & \alpha = 1, \\ \max\{\beta\delta(k-1), \delta_{min}\}, & \alpha = 0, \end{cases} \tag{16}$$

$$y(k) = \begin{cases} \min\{\hat{y}(k), y_{max}\}, & \hat{y}(k) \geq 0. \\ \max\{\hat{y}(k), y_{min}\}, & \hat{y}(k) < 0. \end{cases} \tag{17}$$

where

$$\hat{y}(k) = \hat{x}(k-1) + b(k)\delta(k). \tag{18}$$

In these equations, δ_{min} and δ_{max} are the minimum and maximum step sizes, and y_{min} and y_{max} are the accumulator's negative and positive saturation values, respectively. Over air, the bits are sent in the same order they are generated by the CVSD encoder.

For a 64 kb/s CVSD, the parameters as shown in Table 34 shall be used. The numbers are based on a 16 bit signed number output from the accumulator. These values result in a time constant of 0.5 ms for the accumulator decay, and a time constant of 16 ms for the step size decay.

8.12.3 Error handling

In the DV and HV3 packet, the voice is not protected by FEC. The quality of the voice in an error-prone environment then depends on the robustness of the voice coding scheme. CVSD, in particular, is rather insensitive to random bit errors, which are experienced as white background noise. However, when a packet is rejected because either the channel access code or the HEC test was unsuccessful, measures have to be taken to "fill" in the lost speech segment.

The voice payload in the HV2 packet is protected by a 2/3 rate FEC. For errors that are detected but cannot be corrected, the receiver should try to minimize the audible effects. For instance, from the 15-bit FEC segment with uncorrected errors, the 10-bit information part as found before the FEC decoder should be used. The HV1 packet is protected by a 3-bit repetition FEC. For this code, the decoding scheme will always assume zero or one-bit errors. Thus, there exist no detectable but uncorrectable error events for HV1 packets.

8.12.4 General audio requirements

The following specifications are considered to be default values and shall be supported as a minimum.

Copyright © 2002 IEEE. All rights reserved.

LINKPLAY EXHIBIT 1008
Page 138 of 1169

Table 34—CVSD parameter values[a]

Parameter	Value
h	$1 - \frac{1}{32}$
b	$1 - \frac{1}{1024}$
J	4
K	4
δ_{min}	10
δ_{max}	1280
y_{min}	-2^{15} or $-2^{15} + 1$
y_{max}	$2^{15} - 1$

[a]The values are based on a 16-bit signed number output from the accumulator.

8.12.4.1 Signal levels

For A-law and μ-law log-PCM encoded signals the requirements on signal levels follows ITU-T Recommendation G.711 (11/88).

Full swing at the 16 bit linear PCM interface to the CVSD encoder is defined to be 3 dBm0. A digital CVSD encoded test signal is provided in a Test Signal file available via 2.3. This signal is generated by a software implementation of a reference CVSD encoder. The digital encoder input signal (1020 Hz, sine-wave) generating the test signal has a nominal power of –15 dBm0. When the CVSD encoded test signal is fed through the CVSD receiver chain, the nominal output power should be –15 ± 1.0 dBm0.

8.12.4.2 CVSD audio quality

For Bluetooth audio quality the requirements are put on the transmitter side. The 64 ksamples/s linear PCM input signal must have negligible spectral power density above 4 kHz. A set of reference input signals are encoded by the transmitter and sent through a reference decoder (available via 2.3). The power spectral density in the 4–32 kHz band of the decoded signal at the 64 ksample/s linear PCM output, should be more than 20 dB below the maximum in the 0–4 kHz range.

Copyright © 2002 IEEE. All rights reserved.

8.13 Bluetooth addressing

8.13.1 Bluetooth device address (BD_ADDR)

Each Bluetooth transceiver is allocated a unique 48-bit Bluetooth device address (BD_ADDR). This address is derived from IEEE Std 802-2001. This 48-bit address is divided into three fields:

— LAP field: lower address part consisting of 24 bits

— UAP field: upper address part consisting of 8 bits

— NAP field: non-significant address part consisting of 16 bits

The LAP and UAP form the significant part of the BD_ADDR (see Figure 70). The total address space obtained is 2^{32}.

LSB MSB

company_assigned						company_id					
LAP						UAP		NAP			
0000	0001	0000	0000	0000	0000	0001	0010	0111	1011	0011	0101

Figure 70—Format of BD_ADDR
(company_id should be organizationally unique identifier)

8.13.2 Access codes

In the Bluetooth system, 72-bit and 68-bit access codes are used for signalling purposes. Three different access codes are defined (see also 8.4.2.1):

— device access code (DAC)

— channel access code (CAC)

— inquiry access code (IAC)

There is one general IAC (GIAC) for general inquiry operations and there are 63 dedicated IACs (DIACs) for dedicated inquiry operations. All codes are derived from a LAP of the BD_ADDR. The device access code is used during page, page scan and page response substates. It is a code derived from the unit's BD_ADDR. The channel access code characterizes the channel of the piconet and forms the preamble of all packets exchanged on the channel. The channel access code is derived from the LAP of the master BD_ADDR. Finally, the inquiry access code is used in inquiry operations. A general inquiry access code is common to all Bluetooth units; a set of dedicated inquiry access codes is used to inquire for classes of devices.

The access code is also used to indicate to the receiver the arrival of a packet. It is used for timing synchronization and offset compensation. The receiver correlates against the entire sync word in the access code, providing a very robust signalling. During channel setup, the code itself is used as an ID packet to support the acquisition process. In addition, it is used during random access procedures in the PARK state.

The access code consists of preamble, sync word and a trailer (see 8.4.2). The next two subclauses describe the generation of the sync word.

Copyright © 2002 IEEE. All rights reserved.

LINKPLAY EXHIBIT 1008
Page 140 of 1169

8.13.2.1 Synchronization word definition

The sync words are based on a (64,30) expurgated block code with an overlay (bit-wise XOR) of an 64 bit full length PN-sequence. The expurgated code guarantees large Hamming distance ($d_{min} = 14$) between sync words based on different addresses. The PN sequence improves the auto correlation properties of the access code. The following steps describe how to generate the sync word:

1) Generate information sequence;
2) XOR this with the "information covering" part of the PN overlay sequence;
3) Generate the codeword;
4) XOR the codeword with all 64 bits of the PN overlay sequence;

The information sequence is generated by appending 6 bits to the 24 bit LAP (step 1). The appended bits are 001101 if the MSB of the LAP equals 0. If the MSB of the LAP is 1 the appended bits are 110010. The LAP MSB together with the appended bits constitute a length-seven Barker sequence. The purpose of including a Barker sequence is to further improve the auto correlation properties. In step 2 the information is pre-scrambled by XORing it with the bits $p_{34} \ldots p_{63}$ of the *pseudo-random noise* (PN) sequence (defined in 8.13.2.2). After generating the codeword (step 3), the complete PN sequence is XORed to the codeword (step 4). This step de-scrambles the information part of the codeword. At the same time the parity bits of the codeword are scrambled. Consequently, the original LAP and Barker sequence are ensured a role as a part of the access code sync word, and the cyclic properties of the underlying code is removed. The principle is depicted in Figure 71.

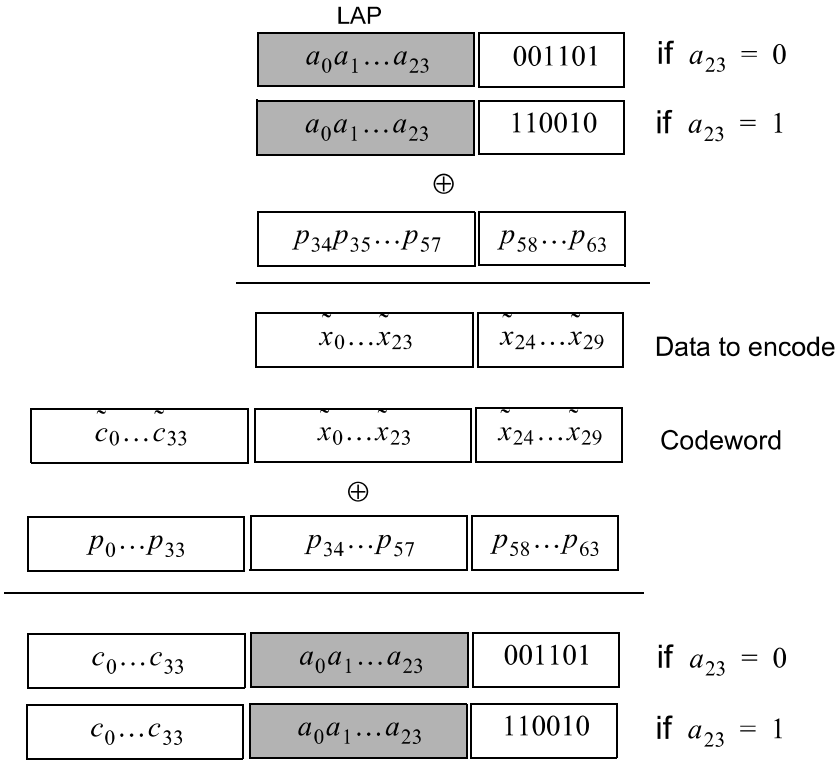


Figure 71—Construction of the sync word

Copyright © 2002 IEEE. All rights reserved.

In the sequel, binary sequences will be denoted by their corresponding D-transform (in which D^i represents a delay of i time units). Let $p'(D) = p'_0 + p'_1 D + \ldots + p'_{62} D^{62}$ be the 63 bit pseudo-random sequence, where p'_0 is the first bit (LSB) leaving the PRNG (see Figure 72), and p'_{62} is the last bit (MSB). To obtain 64 bits, an extra zero is appended at the *end* of this sequence (thus, $p'(D)$ is unchanged). For notational convenience, the reciprocal of this extended polynomial, $p(D) = D^{63} p'(1/D)$, will be used in the sequel. This is the sequence $p'(D)$ in reverse order. We denote the 24 bit lower address part (LAP) of the Bluetooth address by $a(D) = a_0 + a_1 D + \ldots + a_{23} D^{23}$ (a_0 is the LSB of the Bluetooth address).

The (64,30) block code generator polynomial is denoted $g(D) = (1 + D) g'(D)$, where $g'(D)$ is the generator polynomial `157464165547` (octal notation) of a primitive binary (63,30) BCH code. Thus, in octal notation we have

$$g(D) = 260534236651, \tag{19}$$

the left-most bit corresponds to the high-order (g_{34}) coefficient.The dc-free four bit sequences `0101` and `1010` can be written

$$\begin{cases} F_0(D) = D + D^3, \\ F_1(D) = 1 + D^2, \end{cases} \tag{20}$$

respectively. Furthermore, we define

$$\begin{cases} B_0(D) = D^2 + D^3 + D^5, \\ B_1(D) = 1 + D + D^4, \end{cases} \tag{21}$$

which are used to create the length seven Barker sequences. Then, the access code is generated by the following procedure:

1) Format the 30 information bits to encode:

$$x(D) = a(D) + D^{24} B_{a_{23}}(D).$$

2) Add the information covering part of the PN overlay sequence:

$$\tilde{x}(D) = x(D) + p_{34} + p_{35} D + \ldots + p_{63} D^{29}.$$

3) Generate parity bits of the (64,30) expurgated block code:[10]

$$\tilde{c}(D) = D^{34} \tilde{x}(D) \mod g(D).$$

4) Create the codeword:

$$\tilde{s}(D) = D^{34} \tilde{x}(D) + \tilde{c}(D).$$

[10] $x(D)$ mod $y(D)$ denotes the rest when $x(D)$ is divided by $y(D)$.

Copyright © 2002 IEEE. All rights reserved.

LINKPLAY EXHIBIT 1008
Page 142 of 1169

5) Add the PN sequence:

$$s(D) = \tilde{s}(D) + p(D).$$

6) Append the (DC-free) preamble and trailer:

$$y(D) = F_{c_0}(D) + D^4 s(D) + D^{68} F_{a_{23}}(D).$$

8.13.2.2 Pseudo-random noise sequence generation

To generate the pseudo-random noise sequence we use the primitive polynomial $h(D) = 1 + D + D^3 + D^4 + D^6$. The LFSR and its starting state are shown in Figure 72. The PN sequence generated (including the extra terminating zero) becomes (hexadecimal notation) `83848D96BBCC54FC`. The LFSR output starts with the left-most bit of this PN sequence. This corresponds to $p'(D)$ of the previous section. Thus, using the reciprocal $p(D)$ as overlay gives the 64 bit sequence

$$\boldsymbol{p} = 3F2A33DD69B121C1, \tag{22}$$

where the left-most bit is $p_0 = 0$ (there are two initial zeros in the binary representation of the hexadecimal digit 3), and $p_{63} = 1$ is the right-most bit.

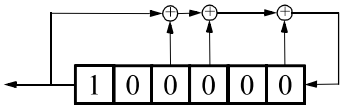


Figure 72—LFSR and the starting state to generate $p'(D)$

8.13.2.3 Reserved addresses for GIAC and DIAC

There is a block of 64 contiguous LAPs reserved for Bluetooth inquiry operations; one LAP common to all Bluetooth devices is reserved for general inquiry, the remaining 63 LAPs are reserved for dedicated inquiry of specific classes of Bluetooth devices. The same 64-block is used regardless of the contents of UAP and NAP. Consequently, none of these LAPs can be part of a user BD_ADDR.

When initializing HEC and CRC for the FHS packet of inquiry response, the UAP is replaced by DCI. Likewise, whenever one of the reserved BD_ADDRs is used for generating a frequency hop sequence, the UAP will be replaced by the DCI.

The reserved LAP addresses are tentatively chosen as `0x9E8B00-0x9E8B3F`. The general inquiry LAP is tentatively chosen to `0x9E8B33`. All addresses have the LSB at the rightmost position, hexadecimal notation.

8.13.3 Active member address (AM_ADDR)

Each slave active in a piconet is assigned a 3-bit active member address (AM_ADDR). The all-zero AM_ADDR is reserved for broadcast messages. The master does not have an AM_ADDR. Its timing relative to the slaves distinguishes it from the slaves. A slave only accepts a packet with a matching AM_ADDR and broadcast packets. The AM_ADDR is carried in the packet header. The AM_ADDR is only valid as long as a slave is active on the channel. As soon as it is disconnected or parked, it loses the AM_ADDR.

Copyright © 2002 IEEE. All rights reserved.

The AM_ADDR is assigned by the master to the slave when the slave is activated. This is either at connection establishment or when the slave is unparked. At connection establishment, the AM_ADDR is carried in the FHS payload (the FHS header itself carries the all-zero AM_ADDR). When unparking, the AM_ADDR is carried in the unpark message.

8.13.4 Parked Member Address (PM_ADDR)

A slave in park mode can be identified by its BD_ADDR or by a dedicated parked member address (PM_ADDR). This latter address is a 8-bit member address that separates the parked slaves. The PM_ADDR is only valid as long as the slave is parked. When the slave is activated it is assigned an AM_ADDR and loses the PM_ADDR. The PM_ADDR is assigned to the slave the moment it is parked.

The all-zero PM_ADDR is reserved for parked slaves that only use their BD_ADDR to be unparked.

8.13.5 Access request address (AR_ADDR)

The access request address is used by the parked slave to determine the slave-to-master half slot in the access window it is allowed to send access request messages in (see also 8.10.8.4.6). The AR_ADDR is assigned to the slave when it enters the park mode and is only valid as long as the slave is parked. The AR_ADDR is not necessarily unique; i.e., different parked slaves may have the same AR_ADDR.

8.14 Bluetooth security

The Bluetooth technology provides peer-to-peer communications over short distances. In order to provide usage protection and information confidentiality, the system has to provide security measures both at the application layer and the link layer. These measures shall be appropriate for a peer environment. This means that in each Bluetooth unit, the authentication and encryption routines are implemented in the same way. Four different entities are used for maintaining security at the link layer: a public address which is unique for each user[11], two secret keys, and a random number which is different for each new transaction. The four entities and their sizes as used in Bluetooth are summarized in Table 35.

Table 35—Entities used in authentication and encryption procedures

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

The Bluetooth device address (BD_ADDR) is the 48-bit IEEE address which is unique for each Bluetooth unit. The Bluetooth addresses are publicly known, and can be obtained via MMI interactions, or, automatically, via an inquiry routine by a Bluetooth unit.

The secret keys are derived during initialization and are further never disclosed. Normally, 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

[11]The BD_ADDR is not a secured identity.

Copyright © 2002 IEEE. All rights reserved.

LINKPLAY EXHIBIT 1008
Page 144 of 1169

between 1 and 16 octets (8–128 bits). The size of the encryption key shall be 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 8.14.5.4). 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 Bluetooth device decides when, or if, to change it. To underline the fundamental importance of the authentication key to a specific Bluetooth link, it will often be referred to as the link key.

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

In the remainder of this chapter, the terms user and application will be used interchangeably to designate the entity that is at the originating or receiving side.

8.14.1 Random number generation

Each Bluetooth unit has a random number generator. Random numbers are used for many purposes within the security functions, for instance, for the challenge-response scheme, for generating authentication and encryption keys, etc. Ideally, a true random generator based on some physical process with inherent randomness is used. Examples of such processes are thermal noise from a semiconductor or resistor and the frequency instability of a free running oscillator. For practical reasons, a software based solution with a pseudo-random generator is probably preferable. In general, it is quite difficult to classify the randomness of a pseudo-random sequence. Within Bluetooth, the requirements placed on the random numbers used are that they be nonrepeating and randomly generated.

The expression “nonrepeating” means that it shall be highly unlikely that the value should repeat itself within the lifetime of the authentication key. For example, a nonrepeating value could be the output of a counter that is unlikely to repeat during the lifetime of the authentication key, or a date/time stamp.

The expression “randomly generated” means that it shall not be possible to predict its value with a chance that is significantly larger than 0 (e.g., greater than $1/2^L$ for a key length of L bits).

Clearly, the LM can use such a generator for various purposes; i.e. whenever a random number is needed (such as the RANDs, the unit keys, K_{init}, K_{master}, and random back-off or waiting intervals).

8.14.2 Key management

It is important that the encryption key size within a specific unit 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 does not accept an encryption key given from higher software layers. Whenever a new encryption key is required, it shall be created as defined in 8.14.5.4.

Changing a link key should 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 8.14.2.2.7.

Copyright © 2002 IEEE. All rights reserved.

8.14.2.1 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 unit is a member of a particular piconet. Thus, the session terminates when the unit disconnects from the piconet.

The link keys are either semipermanent or temporary. A semipermanent link key is stored in nonvolatile memory and may be used after the current session is terminated. Consequently, once a semipermanent link key is defined, it may be used in the authentication of several subsequent connections between the Bluetooth units sharing it. The designation semipermanent is justified by the possibility to change it. How to do this is described in 8.14.2.2.7.

The lifetime of a temporary link key is limited by the lifetime of the current session. It cannot 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) can temporarily replace the current link keys. The details of this procedure are found in 8.14.2.2.6.

In the sequel we sometimes refer to what is denoted as the current link key. This is simply the link key in use at the current moment. It can be semipermanent 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 for different types of applications, four types of link keys have been defined:

— Combination key K_{AB}
— Unit key K_A
— Temporary key K_{master}
— Initialization key K_{init}

In addition to these keys there is an encryption key, denoted K_c. This key is derived from the current link key. Whenever the encryption is activated by a 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.

For a Bluetooth unit, 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 unit A. The unit key is generated once at installation of the Bluetooth unit; thereafter, it is very rarely changed. The combination key is derived from information in both units A and B, and is therefore always dependent on two units. The combination key is derived for each new combination of two Bluetooth units.

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

Copyright © 2002 IEEE. All rights reserved.

LINKPLAY EXHIBIT 1008
Page 146 of 1169

The master key, K_{master}, is a link key only used during the current session. It will replace the original link key only temporarily. For example, this may be utilized when a master wants to reach more than two Bluetooth units simultaneously using the same encryption key (see 8.14.2.2.6).

The initialization key, K_{init}, is used as 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 is only to be used during initialization.

The PIN can be a fixed number provided with the Bluetooth unit (for example when there is no MMI as in a PSTN plug). Alternatively, the PIN can be selected arbitrarily by the user, and then entered in both units that have to be matched. The latter procedure is used when both units have an MMI, for example a phone and a laptop. Entering a PIN in both units is more secure than using a fixed PIN in one of the units, and should be used whenever possible. Even if a fixed PIN is used, it shall be possible to change the PIN; this in order to prevent re-initialization by users who once got hold of the PIN. If no PIN is available, a default value of zero is to be used.

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 can be chosen to be any length from 1 to 16 octets. For the longer lengths, we envision the units exchanging PIN codes not through mechanical (i.e. human) interaction, but rather through means supported by 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 units, just as in the case of a shorter PIN code.

8.14.2.2 Key generation and initialization

The link keys have to be generated and distributed among the Bluetooth units in order to be used in the authentication procedure. Since the link keys must be secret, they cannot be obtained through an inquiry routine in the same way as the Bluetooth addresses. The exchange of the keys takes place during an initialization phase which has to be carried out separately for each two units that want to implement authentication and encryption. All initialization procedures consist of the following five parts:

- Generation of an initialization key
- Generation of link key
- Link key exchange
- Authentication
- Generating of encryption key in each unit (optional)

After the initialization procedure, the units can proceed to communicate, or the link can be disconnected. If encryption is implemented, the E_0 algorithm is used with the proper encryption key derived from the current link key. For any new connection established between units A and B, they will 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 will be created next time encryption is activated.

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

8.14.2.2.1 Generation of initialization key, K_{init}

A link key used temporarily during initialization is derived – the initialization key K_{init}. This key is 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 87. The 128-bit output from E_{22} will be used for

Copyright © 2002 IEEE. All rights reserved.
LINKPLAY EXHIBIT 1008
Page 147 of 1169

key exchange during the generation of a link key. When the units 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 unit has a fixed PIN the BD_ADDR of the other unit is used. If both units have a variable PIN the BD_ADDR of the device that received IN_RAND is used. If both units 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 unit with a variable PIN. A fraudulent Bluetooth unit may try to test a large number of PINs by each time claiming another BD_ADDR. It is the application's responsibility to take countermeasures against this threat. If the device address is kept fixed, the waiting interval until next try is permitted is increased exponentially (see 8.14.4.1).

The details of the E_{22} algorithm can be found in 8.14.5.3.

8.14.2.2.2 Authentication

The authentication procedure is carried out as described in 8.14.4. Note that during each authentication, a new AU_RAND_A is issued.

Mutual authentication is achieved by first performing the authentication procedure in one direction and immediately followed by 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 is used for ciphering key generation as described in 8.14.2.2.5.

The claimant/verifier status is determined by the LM.

8.14.2.2.3 Generation of a unit key

A unit key K_A is generated when the Bluetooth unit is for the first time in operation; i.e. not during each initialization! The unit key is generated by the E_{21} algorithm as described in 8.14.5.3. Once created, the unit key is stored in non-volatile memory and (almost) never changed. If after initialization the unit key is changed, the previously initialized units will possess a wrong link key. At initialization, the application has to determine which of the two parties will provide the unit key as link key. Typically, this will be the unit with restricted memory capabilities, since this unit only has to remember its own unit key. The unit key is transferred to the other party and then stored as link key for that particular party. So, for example in Figure 73, the unit key of unit A, K_A, is being used as link key for the connection A-B; unit A sends the unit key K_A to unit B; unit B will store K_A as the link key K_{BA}. When the unit key has been exchanged, the initialization key shall be discarded in both units. For another initialization, for example with unit C, unit A will reuse its unit key K_A, whereas unit C stores it as K_{CA}.

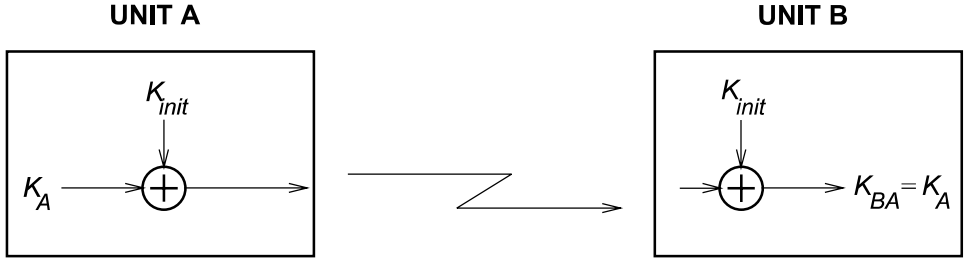


Figure 73—Generation of unit key

Copyright © 2002 IEEE. All rights reserved.

8.14.2.2.4 Generation of a combination key

If it is desired to use a combination key, this key is first generated during the initialization procedure. The combination key is the combination of two numbers generated in unit A and B, respectively. First, each unit generates a random number, say LK_RAND_A and LK_RAND_B. Then, utilizing E_{21} with the random number and the own BD_ADDR, the two random numbers

$$LK\_K_A = E_{21}(LK\_RAND_A, BD\_ADDR_A), \tag{23}$$

and

$$LK\_K_B = E_{21}(LK\_RAND_B, BD\_ADDR_B), \tag{24}$$

are created in unit A and unit B, respectively. These numbers constitute the units' contribution to the combination key that is to be created. Then, the two random numbers LK_RAND_A and LK_RAND_B are exchanged securely by XORing with the current link key, say K. Thus, unit A sends $K \oplus LK\_RAND_A$ to unit B, while unit B sends $K \oplus LK\_RAND_B$ to unit A. Clearly, 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 unit recalculates the other units contribution to the combination key. This is possible since each unit knows the Bluetooth device address of the other unit. Thus, unit A calculates Equation (24) and unit B calculates Equation (23). After this, both units 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 is stored in unit A as the link key K_{AB} and in unit B as the link key K_{BA}. When both units 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 74. The old link key (K) shall be discarded after the exchange of a new combination key has succeeded.

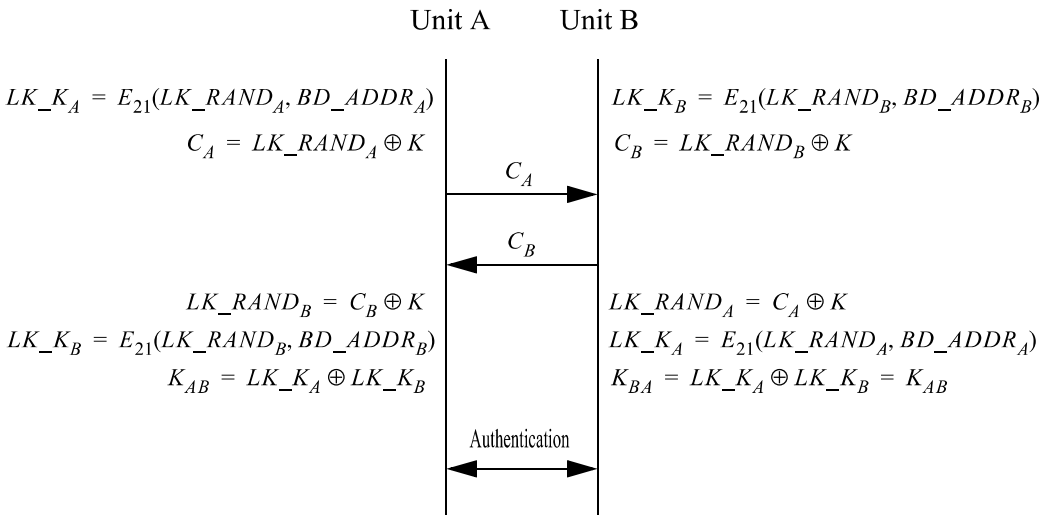


Figure 74—Generating a combination key

Copyright © 2002 IEEE. All rights reserved.
LINKPLAY EXHIBIT 1008
Page 149 of 1169

8.14.2.2.5 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 is derived from the master BD_ADDR. Otherwise the value of COF is set to the value of ACO as computed during the authentication procedure. More precisely, we have[12]

$$\mathrm{COF} = \begin{cases} \mathrm{BD\_ADDR} \cup \mathrm{BD\_ADDR}, & \text{if link key is a master key} \\ \mathrm{ACO}, & \text{otherwise.} \end{cases} \tag{25}$$

There is an explicit call of E_3 when the LM activates encryption. Consequently, the encryption key is automatically changed each time the unit enters the encryption mode. The details of the key generating function E_3 can be found in 8.14.5.4.

8.14.2.2.6 Point-to-multipoint configuration

It is quite 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 may cause unwanted capacity loss for the piconet. Moreover, a Bluetooth unit (slave) is not capable of switching between two or more encryption keys in real time (e.g., after looking at the AM_ADDR in the header). Thus, the master cannot use different encryption keys for broadcast messages and individually addressed traffic. Alternatively, the master may tell several slave units to use a common link key (and, hence, indirectly also to use a common encryption key) and broadcast the information encrypted. For many applications, this key is only of temporary interest. In the sequel, this key is denoted K_{master}.

The transfer of necessary parameters is protected by the routine described in 8.14.2.2.8. 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 also has to generate and distribute a common EN_RAND to all participating slaves. Using this random number and the newly derived master key, each slave generates a new encryption key.

Note that the master must negotiate what encryption key length to use individually with each slave who wants to use the master key. In case the master already has negotiated with some of these slaves, it has knowledge of what sizes can be accepted. Clearly, there might be situations where the permitted key lengths of some units are incompatible. In that case, the master must have the limiting unit excluded 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. Clearly, each slave in possession of the master key can eavesdrop on all encrypted traffic, not only the traffic intended for itself. If so desired, the master can tell all participants to fall back to their old link keys simultaneously.

8.14.2.2.7 Modifying the link keys

In certain circumstances, it is desirable to be able to modify the link keys. A link key based on a unit key can be changed, but not very easily. The unit key is created once during the first use. Changing the unit key is a less desirable alternative, as several units may share the same unit key as link key (e.g., a printer whose unit key is distributed among all users using the printer's unit key as link key). Changing the unit key will require re-initialization of all units trying to connect. In certain cases, this might be desirable; for example, to deny access to previously allowed units.

[12] $x \cup y$ denotes the concatenation of the octet strings x and y.

Copyright © 2002 IEEE. All rights reserved.

LINKPLAY EXHIBIT 1008
Page 150 of 1169

If the key change concerns combination keys, then the procedure is rather straightforward. The change procedure is identical to the procedure illustrated in Figure 74, 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. In fact, 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.

Of course, starting up an entirely new initialization procedure is also a possibility. In that case, user interaction is necessary since a PIN is required in the authentication and encryption procedures.

8.14.2.2.8 Generating a master key

The key-change routines described in 8.14.2.2.1 through 8.14.2.2.7 are semipermanent. To create the master link key, which can replace the current link key during an initiated session (see 8.14.2.2.6), other means are needed. First, the master creates a new link key from two 128-bit random numbers, RAND1 and RAND2. This is done by

$$K_{\text{master}} = E_{22}(\text{RAND1}, \text{RAND2}, 16). \quad (26)$$

Clearly, this key is a 128-bit random number. The reason to use the output of E_{22} and not directly chose 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 Bluetooth unit.

Then, a third random number, e.g., RAND, is transmitted to the slave. Using E_{22} with the current link key and RAND as inputs, both the master and slave computes a 128-bit overlay. The master sends the bitwise XOR of the overlay and the new link key to the slave. The slave, who knows the overlay, recalculates K_{master}. To confirm the success of this transaction, the units shall perform a mutual authentication procedure using the new link key. This procedure is then repeated for each slave who shall receive the new link key. The ACO values from the involved authentications should not replace the current existing ACO as this ACO is needed to (re)compute a ciphering key when the master wants to fall back to the previous link (non-temporary) key.

When so required, and potentially long after the actual distribution of the master key, the master activates encryption by an LM command. Before doing that, the master shall ensure that all slaves receive the same random number, say EN_RAND, since the encryption key is derived through the means of E_3 individually in all participating units. Then, each slave computes a new encryption key,

$$K_C = E_3(K_{\text{master}}, \text{EN\_RAND}, \text{COF}), \quad (27)$$

where the value of COF is derived from the master's BD_ADDR as specified by Equation (25). The details on the encryption key generating function can be found in 8.14.5.4. The principle of the message flow between the master and slave when generating the master key is depicted in Figure 75. Note that in this case the ACO produced during the authentication is not used when computing the ciphering key.

Copyright © 2002 IEEE. All rights reserved.
LINKPLAY EXHIBIT 1008
Page 151 of 1169

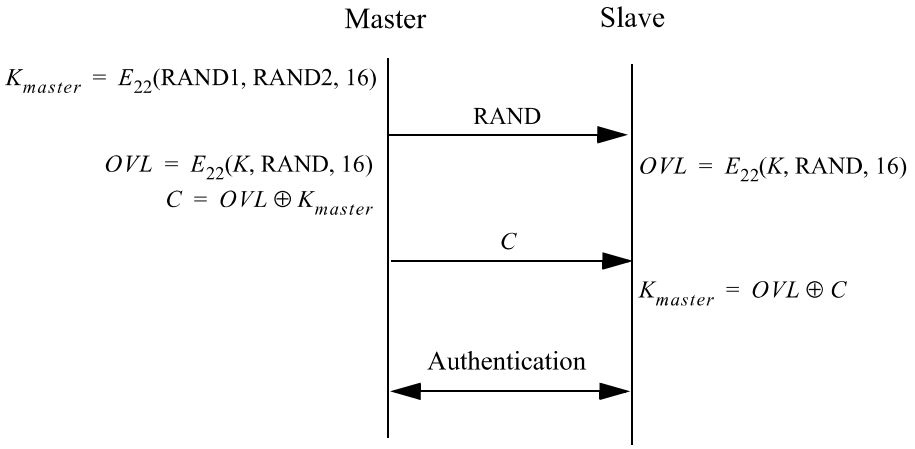


Figure 75—Master link key distribution and computation of the corresponding encryption key

8.14.3 Encryption

User information can be protected by encryption of the packet payload; the access code and the packet header are never encrypted. The encryption of the payloads is carried out with a stream cipher called E_0 that is re-synchronized for every payload. The overall principle is shown in Figure 76.

The stream cipher system E_0 consists of three parts. One part performs the initialization (generation of the payload key), the second part generates the key stream bits, and the third part performs the encryption and decryption. The payload key generator is very simple, it merely combines the input bits in an appropriate order and shift them into the four LFSRs used in the key stream generator. The main part of the cipher system is the second, as it also will be used for the initialization. The key stream bits are generated by a method derived from the summation stream cipher generator attributable to Massey and Rueppel [B12]. The 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 so-called correlation attacks, the high re-synchronization frequency will disrupt such attacks.

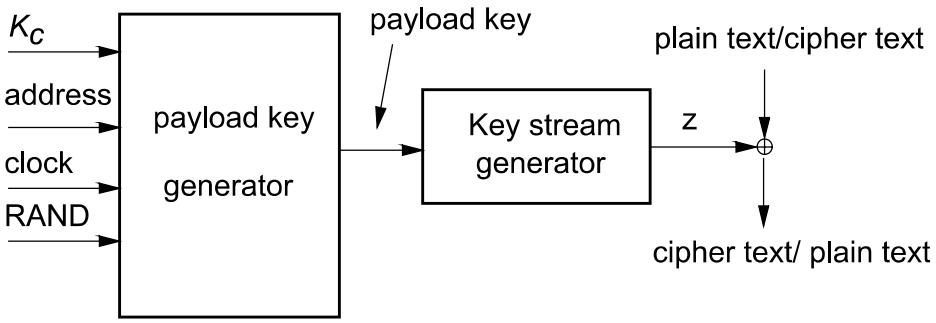


Figure 76—Stream ciphering for Bluetooth with E_0

Copyright © 2002 IEEE. All rights reserved.

8.14.3.1 Encryption key size negotiation

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

The master sends a suggested value, $L_{sug}^{(M)}$, to the slave. Initially, the suggested value is set to $L_{max}^{(M)}$. If $L_{min}^{(S)} \le L_{sug}^{(M)}$, and, the slave supports the suggested length, the slave acknowledges, and this value will be the length of the encryption key for this link. However, if both conditions are not fulfilled, the slave sends a new proposal, $L_{sug}^{(S)} < L_{sug}^{(M)}$, to the master. This value should be the largest among all supported lengths less than the previous master suggestion. Then, the master performs the corresponding test on the slave suggestion. This procedure is repeated until a key length agreement is reached, or, one unit aborts the negotiation. An abortion 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 units. In case of abortion, Bluetooth link encryption can not 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 unit could enforce a weak protection on a link by claiming a small maximum key size.

8.14.3.2 Encryption modes

If a slave has a semipermanent link key (i.e., a combination key or a unit key), it can only accept encryption on slots individually addressed to itself (and, of course, in the reverse direction to the master). In particular, it will assume that broadcast messages are not encrypted. The possible traffic modes are described in Table 36. When an entry in the table refers to a link key, it means that the encryption/decryption engine uses the encryption key derived from that link key.

Table 36—Possible traffic modes for a slave using a semipermanent link key

Broadcast traffic	Individually addressed traffic
No encryption	No encryption
No encryption	Encryption, Semi-permanent link key

If the slave has received a master key, there are three possible combinations, as defined in Table 37. In this case, all units in the piconet uses a common link key, K_{master}. Since the master uses encryption keys derived from this link key for all secure traffic on the piconet, it is possible to avoid ambiguity in the participating slaves on which encryption key to use. Also in this case, the default mode is that broadcast messages are not encrypted. A specific LM-command is required to activate encryption, both for broadcast and for individually addressed traffic.

The master can issue an LM-command to the slaves telling them to fall back to their previous semipermanent link key. Then, regardless of the previous mode they were in, they will end up in the first row of Table 36; i.e., no encryption.

Copyright © 2002 IEEE. All rights reserved.

Table 37—Possible encryption modes for a slave in possession of a master key

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

8.14.3.3 Encryption concept

For the encryption routine, a stream cipher algorithm will be 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 is ciphered separately. The cipher algorithm E_0 uses the master Bluetooth address, 26 bits of the master real-time clock ($CLK_{26\text{-}1}$) and the encryption key K_C as input, see Figure 77 (where it is assumed that unit 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 8.14.5.4). The random number is 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 is 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 8.14.3.5.

The real-time clock is incremented for each slot. The E_0 algorithm is 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 is being used for the entire packet.

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

Copyright © 2002 IEEE. All rights reserved.

LINKPLAY EXHIBIT 1008
Page 154 of 1169

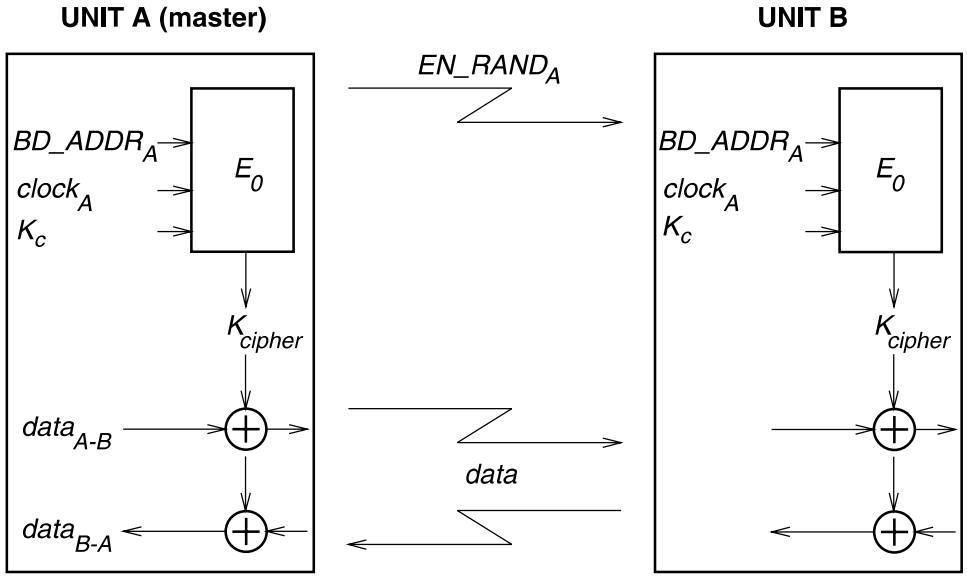


Figure 77—Functional description of the encryption procedure

8.14.3.4 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 is presented with an encryption key K_C, an 48-bit Bluetooth address, the master clock bits $CLK_{26\text{-}1}$, and a 128-bit RAND value. Figure 78 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 38. 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 realization and obtaining good statistical properties of the generated sequences.

Copyright © 2002 IEEE. All rights reserved.

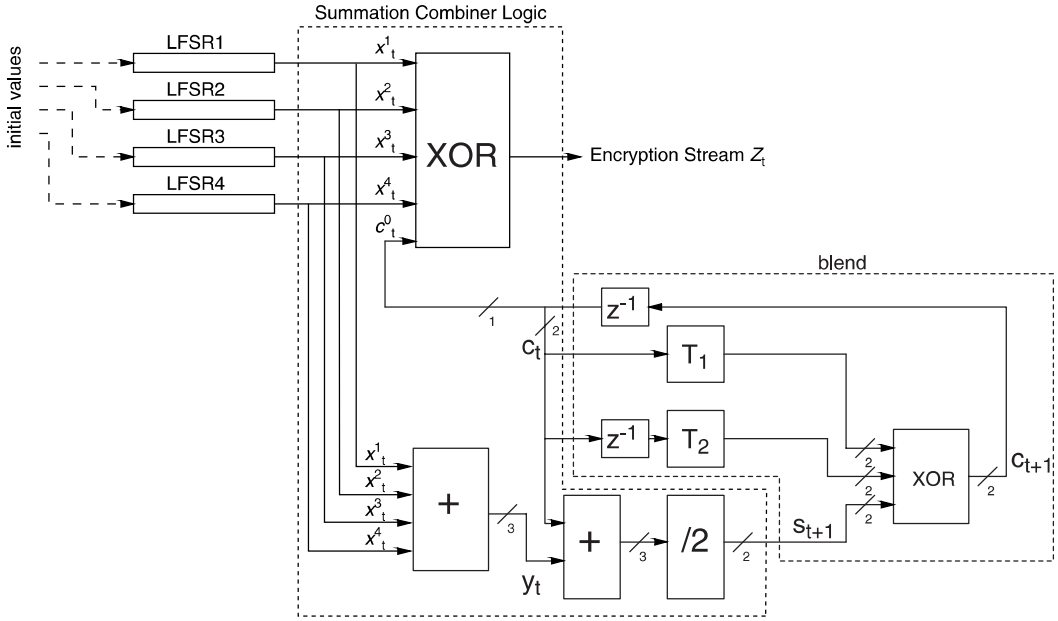


Figure 78—Concept of the encryption engine

Table 38—The four primitive feedback polynomials

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

Let x_t^i denote the t^{th} symbol of LSFR_i. From the four-tuple $x_t^1, \ldots, x_t^4$ we derive the value y_t as follows:

$$y_t = \sum_{i=1}^{4} x_t^i, \tag{28}$$

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 now given by the following equations:

$$z_t = x_t^1 \oplus x_t^2 \oplus x_t^3 \oplus x_t^4 \oplus c_t^0 \in \{0, 1\}, \tag{29}$$

Copyright © 2002 IEEE. All rights reserved.

LINKPLAY EXHIBIT 1008
Page 156 of 1169

$$s_{t+1} = (s^1_{t+1}, s^0_{t+1}) = \left\lfloor \frac{y_t + c_t}{2} \right\rfloor \in \{0, 1, 2, 3\}, \tag{30}$$

$$c_{t+1} = (c^1_{t+1}, c^0_{t+1}) = s_{t+1} \oplus T_1[c_t] \oplus T_2[c_{t-1}], \tag{31}$$

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 α be a zero of this polynomial in GF(4). The mappings T_1 and T_2 are now defined as follows:

$$T_1: \text{GF(4)} \rightarrow \text{GF(4)}$$

$$x \mapsto x$$

$$T_2: \text{GF(4)} \rightarrow \text{GF(4)}$$

$$x \mapsto (\alpha + 1)x.$$

Table 39 summarizes the elements of GF(4) written as binary vectors.

Table 39—Mappings T_1 and T_2

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

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

$$T_1: \quad (x_1, x_0) \mapsto (x_1, x_0),$$

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

8.14.3.4.1 The operation of the cipher

Figure 79 gives an overview of the operation in time. The encryption algorithm shall run through the initialization phase before the start of transmitting or receiving a new packet. Thus, for multislot packets the cipher is initialized using the clock value of the first slot in the multislot sequence.

Copyright © 2002 IEEE. All rights reserved.

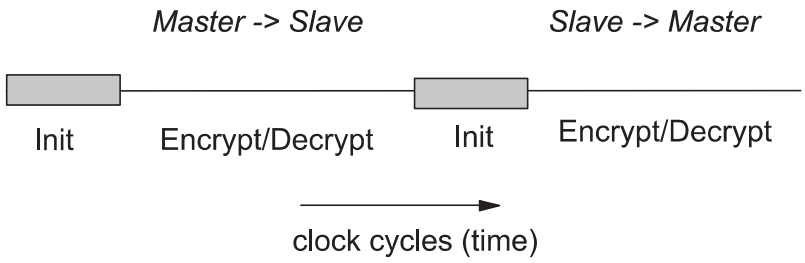


Figure 79—Overview of the operation of the encryption engine. Between each start of a packet (TX or RX), the LFSRs are re-initialized

8.14.3.5 LFSR initialization

The key stream generator needs to be 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 itself. The input parameters are the key K_C, a 128-bit random number RAND, a 48-bit Bluetooth address, and the 26 master clock bits $CLK_{26\text{-}1}$.

The effective length of the encryption key can 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 is 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 is defined in Equation (32).

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 denoted by the notation $X[i]$. We define bit 0 of X to be 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 address is the LSB of ADR[3].

The details of the initialization are 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 will be denoted K'_C:

$$K'_C(x) = g_2^{(L)}(x)(K_C(x) \mod g_1^{(L)}(x)), \tag{32}$$

where $\deg(g_1^{(L)}(x)) = 8L$ and $\deg(g_2^{(L)}(x)) \le 128 - 8L$. The polynomials are defined in Table 40.

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

   i) Open all switches shown in Figure 80.

   ii) Arrange inputs bits as shown in Figure 80; Set the content of all shift register elements to zero. Set $t = 0$.

Copyright © 2002 IEEE. All rights reserved.

LINKPLAY EXHIBIT 1008
Page 158 of 1169

iii) Start shifting bits into the LFSRs. The rightmost bit at each level of Figure 80 is the first bit to enter the corresponding LFSR.

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

v) 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, from this moment forward their content will be used in computing the output sequence.

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

Note that 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.

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

8) 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 81 at $t = 240$.

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

Table 40—Polynomials used when creating K'_C [a]

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
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

[a]All polynomials are in hexadecimal notation. The LSB is in the rightmost position.

Copyright © 2002 IEEE. All rights reserved.
LINKPLAY EXHIBIT 1008
Page 159 of 1169

In Figure 80, all bits are shifted into the LFSRs, starting with the least significant bit (LSB). For instance, from the third octet of the address, ADR[2], first ADR_{16} is entered, followed by ADR_{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).

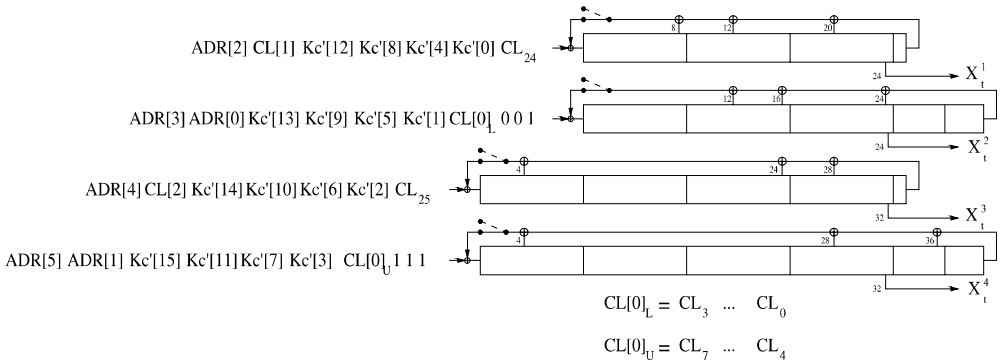


Figure 80—Arranging the input to the LFSRs

In Figure 81, 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 latest 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 are 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 $LFSR_4$.

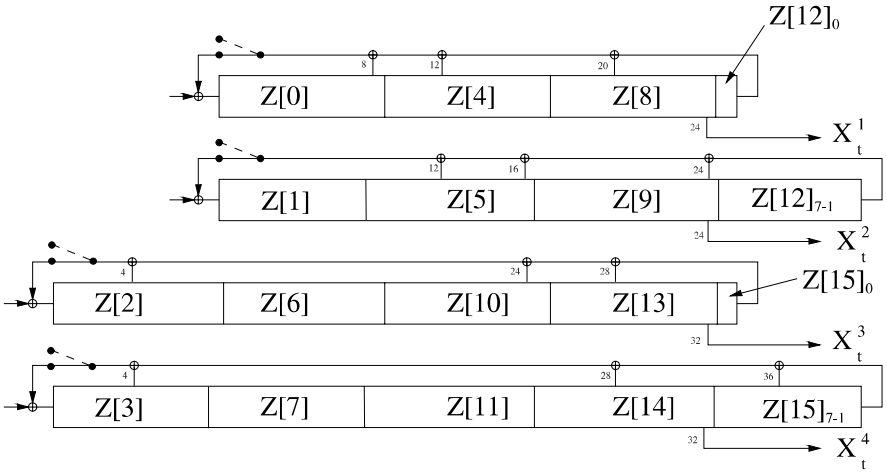


Figure 81—Distribution of the 128 last generated output symbols within the LFSRs

Copyright © 2002 IEEE. All rights reserved.

LINKPLAY EXHIBIT 1008
Page 160 of 1169

8.14.3.6 Key stream sequence

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

Sample data of the encryption output sequence can be found in the 2.4.1, Encryption Sample Data. A necessary, but not sufficient, condition for all Bluetooth-compliant implementations is to produce these encryption streams for identical initialization values.

8.14.4 Authentication

Authentication in a Bluetooth piconet uses a challenge-response scheme in which a claimant's knowledge of a secret key is checked through a two-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 82). This figure shows also that in Bluetooth 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.[13] The secret K shared by units A and B is the current link key.

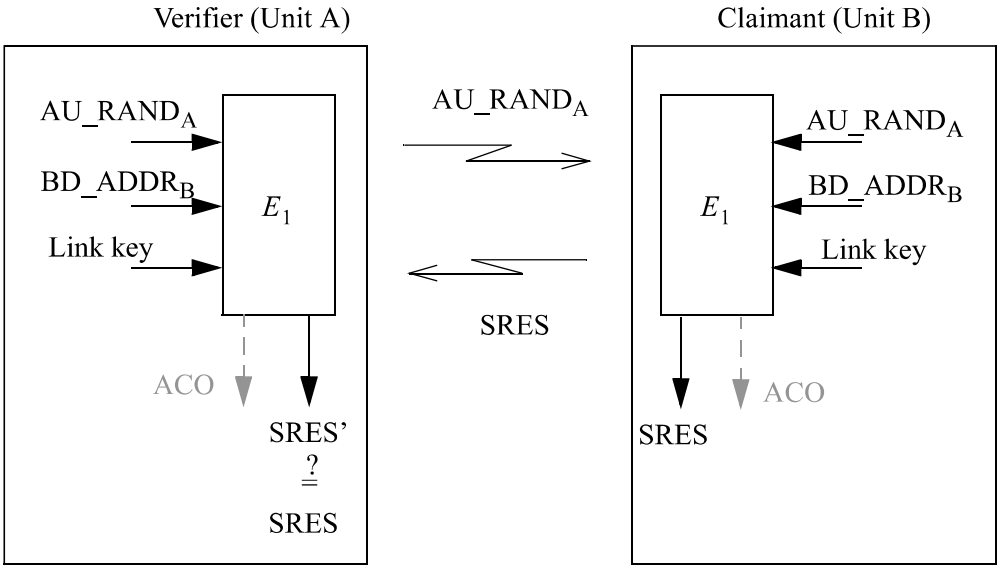


Figure 82—Challenge-response for the Bluetooth

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

Copyright © 2002 IEEE. All rights reserved.

The challenge-response scheme for symmetric keys used in the Bluetooth is depicted in Figure 83.

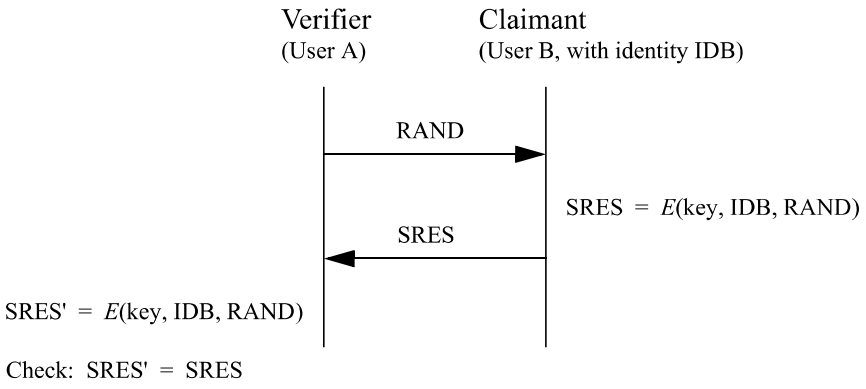


Figure 83—Challenge-response for symmetric key systems

In Bluetooth authentication, the verifier is not necessarily the master. The application indicates who has to be authenticated by whom. Certain applications only require a one-way authentication. However, in some peer-to-peer communications, one might prefer a mutual authentication in which each unit is subsequently the challenger (verifier) in two authentication procedures. The LM coordinates the indicated authentication preferences by the application to determine in which direction(s) the authentication(s) has to take place. For mutual authentication with the units of Figure 82, after unit A has successfully authenticated unit B, unit B could authenticate unit A by sending a AU_RAND_B (different from the AU_RAND_A that unit A issued) to unit A, and deriving the SRES and SRES' from the new AU_RAND_B, the address of unit A, and the link key K_{AB}.

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

8.14.4.1 Repeated attempts

When the authentication attempt fails, a certain 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 unit claiming the same identity as the suspicious unit. For each subsequent authentication failure with the same Bluetooth address, the waiting interval shall be increased exponentially. That is, after each failure, the waiting interval before a new attempt can be made, for example, twice as long as the waiting interval prior to the previous attempt[14]. 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 being made during a certain time period. This procedure prevents an intruder from repeating the authentication procedure with a large number of different keys.

To make the system somewhat less vulnerable to denial-of-service attacks, the Bluetooth units should keep a list of individual waiting intervals for each unit it has established contact with. Clearly, the size of this list must be restricted only to contain the *N* units with which the most recent contact has been made. The number *N* can vary for different units depending on available memory size and user environment.

[14]Another appropriate value larger than 1 may be used.

Copyright © 2002 IEEE. All rights reserved.

LINKPLAY EXHIBIT 1008
Page 162 of 1169

8.14.5 The authentication and key-generating functions

This subclause describes the algorithmic means for supporting the Bluetooth security requirements on authentication and key generation.

8.14.5.1 The authentication function E_1

The authentication function E_1 proposed for the Bluetooth is a computationally secure authentication code, or often called a MAC. E_1 uses the encryption function called SAFER+. The algorithm is an enhanced version of an existing 64-bit block cipher SAFER-SK128, and it is freely available. In the sequel, the block cipher will be denoted as the function A_r which maps under a 128-bit key, a 128-bit input to a 128-bit output, i.e.,

$$\begin{aligned} A_r\colon\ \{0,1\}^{128}\times\{0,1\}^{128} &\rightarrow \{0,1\}^{128} \\ (k\times x) &\mapsto t. \end{aligned} \tag{33}$$

The details of A_r are given in 8.14.5.2. The function E_1 is constructed using A_r as follows:

$$\begin{aligned} E_1\colon\ \{0,1\}^{128}\times\{0,1\}^{128}\times\{0,1\}^{48} &\rightarrow \{0,1\}^{32}\times\{0,1\}^{96} \\ (K,\ \text{RAND},\ \text{address}) &\mapsto (\text{SRES},\ \text{ACO}), \end{aligned} \tag{34}$$

where $\text{SRES} = Hash(K, \text{RAND}, \text{address}, 6)[0, \ldots, 3]$, where $Hash$ is a keyed hash function defined as[15]

$$\begin{aligned} &Hash\colon\{0,1\}^{128}\times\{0,1\}^{128}\times\{0,1\}^{8\times L}\times\{6,12\}\rightarrow\{0,1\}^{128} \\ &(K, I_1, I_2, L)\mapsto A'_r([\tilde{K}], [E(I_2, L)+_{16}(A_r(K, I_1)\oplus_{16} I_1)]), \end{aligned} \tag{35}$$

and where

$$\begin{aligned} &E\colon\{0,1\}^{8\times L}\times\{6,12\}\rightarrow\{0,1\}^{8\times 16} \\ &(X[0,\ldots,L-1], L)\mapsto (X[i(\text{mod L})]\ \text{ for } i=0...15), \end{aligned} \tag{36}$$

is an expansion of the L octet word X into a 128-bit word. Thus we see that we have to evaluate the function A_r twice for each evaluation of E_1. The key K for the second use of A_r (actually A'_r) is offseted from K as follows:[16]

[15]The operator $+_{16}$ denotes bytewise addition mod 256 of the 16 octets, and the operator $\oplus_{16}$ denotes bytewise X-raying of the 16 octets.

[16]The constants are the first largest primes below 257 for which 10 is a primitive root.

Copyright © 2002 IEEE. All rights reserved.
LINKPLAY EXHIBIT 1008
Page 163 of 1169

$$
\begin{aligned}
\tilde{K}[0] &= (\tilde{K}[0] + 233) \mod 256, & \tilde{K}[1] &= \tilde{K}[1] \oplus 229,\\
\tilde{K}[2] &= (\tilde{K}[2] + 223) \mod 256, & \tilde{K}[3] &= \tilde{K}[3] \oplus 193,\\
\tilde{K}[4] &= (\tilde{K}[4] + 179) \mod 256, & \tilde{K}[5] &= \tilde{K}[5] \oplus 167,\\
\tilde{K}[6] &= (\tilde{K}[6] + 149) \mod 256, & \tilde{K}[7] &= \tilde{K}[7] \oplus 131,\\
\tilde{K}[8] &= \tilde{K}[8] \oplus 233, & \tilde{K}[9] &= (\tilde{K}[9] + 229) \mod 256,\\
\tilde{K}[10] &= \tilde{K}[10] \oplus 223, & \tilde{K}[11] &= (\tilde{K}[11] + 193) \mod 256,\\
\tilde{K}[12] &= \tilde{K}[12] \oplus 179, & \tilde{K}[13] &= (\tilde{K}[13] + 167) \mod 256,\\
\tilde{K}[14] &= \tilde{K}[14] \oplus 149, & \tilde{K}[15] &= (\tilde{K}[15] + 131) \mod 256.
\end{aligned}
\tag{37}
$$

A data flowchart of the computation of E_1 is depicted in Figure 84. 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 Equation (25) and Equation (45)]. The value of ACO is formed by the octets 4 through 15 of the output of the hash function defined in Equation (35), i.e.,

$$\text{ACO} = Hash(K, \text{RAND}, \text{address}, 6)[4, \ldots, 15]. \tag{38}$$

Copyright © 2002 IEEE. All rights reserved.

LINKPLAY EXHIBIT 1008
Page 164 of 1169

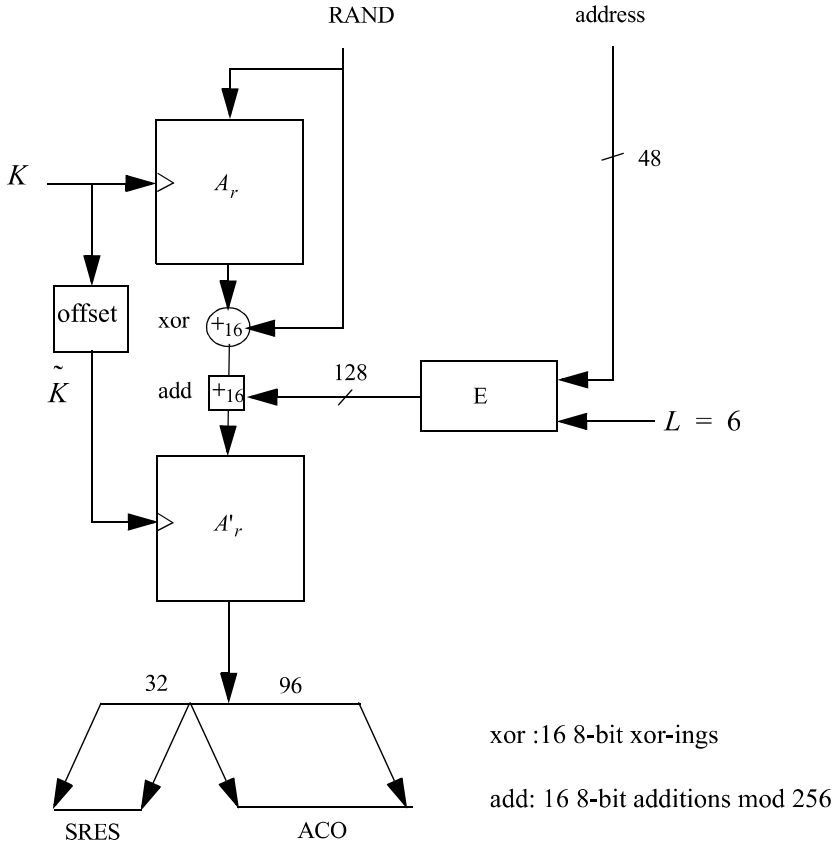


Figure 84—Flow of data for the computation of E_1

8.14.5.2 The functions A_r and A'_r

The function A_r is identical to SAFER+. It consists of a set of eight layers, (each layer is called a round) and a parallel mechanism for generating the sub keys $K_p[j]$, $p = 1, 2, \ldots, 17$, the so-called 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 the 3rd round. 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 85 for details.

8.14.5.2.1 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 are shown in Figure 85. The permutation boxes show how input byte indices are mapped onto output byte indices. Thus, position 0 (leftmost) is mapped on position 8, position 1 is mapped on position 11, etc. The sub keys for round $r, r = 1, 2, \ldots, 8$ are denoted $K_{2r-1}[j]$, $K_{2r}[j]$, $= 0, 1, \ldots, 15$. After the last round $k_{17}[j]$ is applied in a similar fashion as all previous odd numbered keys.

Copyright © 2002 IEEE. All rights reserved.

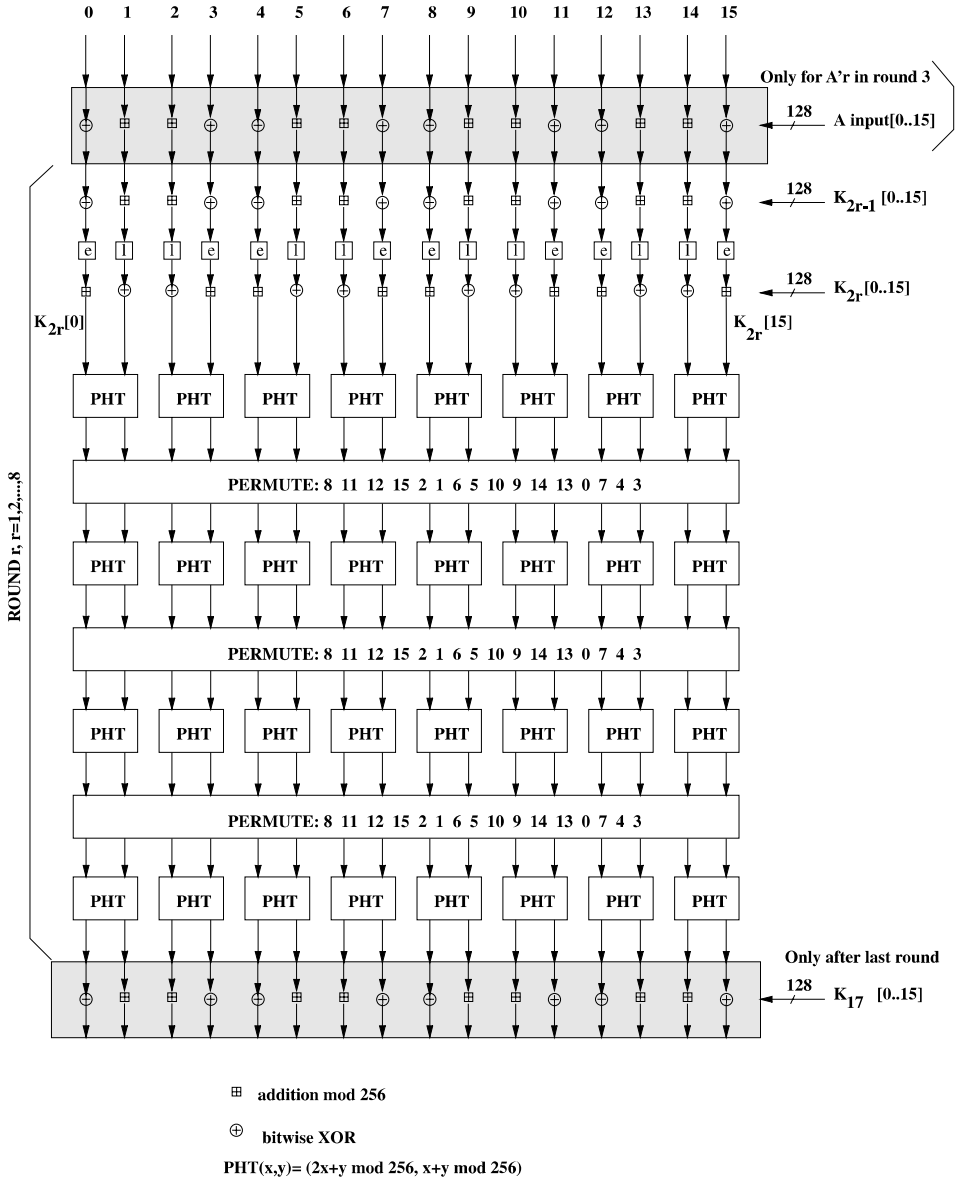


Figure 85—One round in A_r and A'_r

8.14.5.2.2 The substitution boxes “e” and “l”

In Figure 85 two boxes occur, marked “e” and “l”. These boxes implement the same substitutions as used in SAFER+; i.e., they implement the following:

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

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

Copyright © 2002 IEEE. All rights reserved.

LINKPLAY EXHIBIT 1008
Page 166 of 1169

8.14.5.2.3 Key scheduling

In each round, two batches of 16 octet-wide keys are needed. These so-called round keys are derived as specified by the key scheduling in SAFER+. Figure 86 gives an overview of how the round keys $K_p[j]$ are determined. The bias vectors B_2, B_3, ..., B_{17} are computed according to following equation:

$$B_p[i] = ((45^{(45^{17p+i+1} \bmod 257)} \bmod 257) \bmod 256), \text{ for } i = 0, \ldots, 15. \quad (39)$$

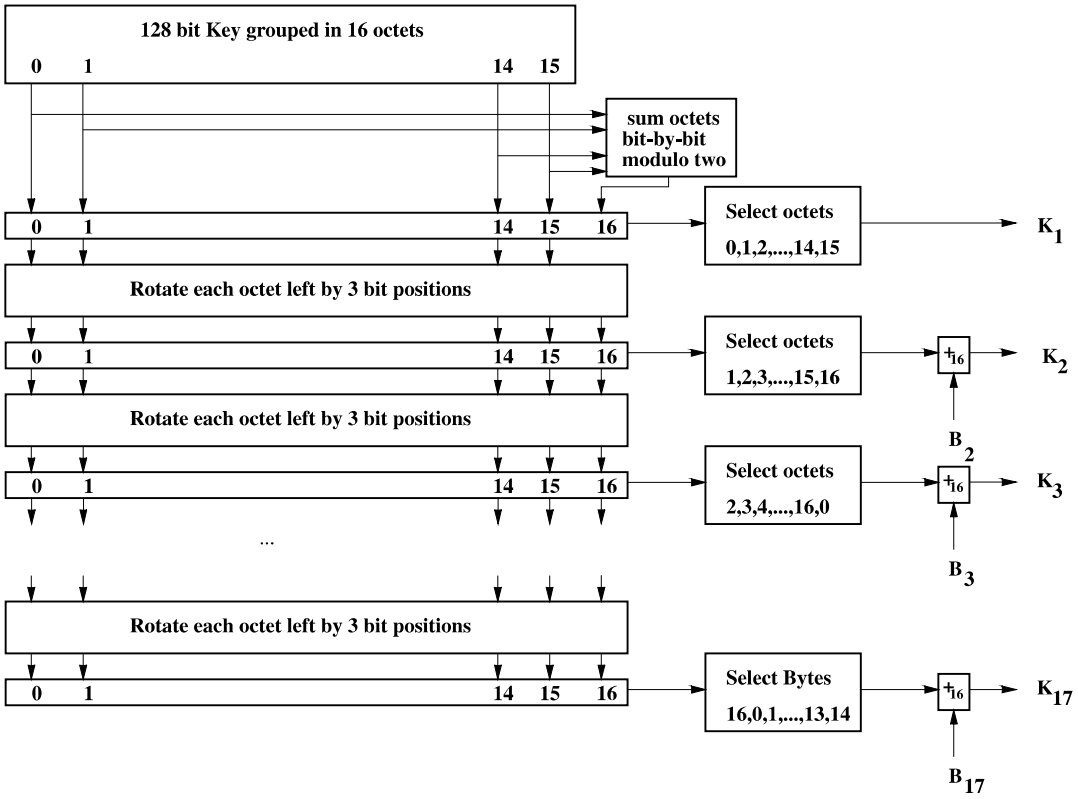


Figure 86—Key scheduling in A_r

8.14.5.3 E_2-key generation function for authentication

The key used for authentication is derived through a procedure that is shown in Figure 87. The figure shows two different modes of operation for the algorithm. In the first mode, the function E_2 should produce on input of a 128-bit RAND value and a 48-bit address, a 128-bit link key K. This mode is utilized when creating unit keys and combination keys. In the second mode the function E_2 should produce, on input of a 128-bit RAND value and an L octet user PIN, a 128-bit link key K. The second mode is used to create the initialization key, and also whenever a master key is to be generated.

NOTE—Mode 1 is used for unit and combination keys, while mode 2 is used for K_{init} and K_{master}.

Copyright © 2002 IEEE. All rights reserved.

When the initialization key is generated, the PIN is augmented with the BD_ADDR (see 8.14.2.2.1 for which address to use). The augmentation always starts 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.

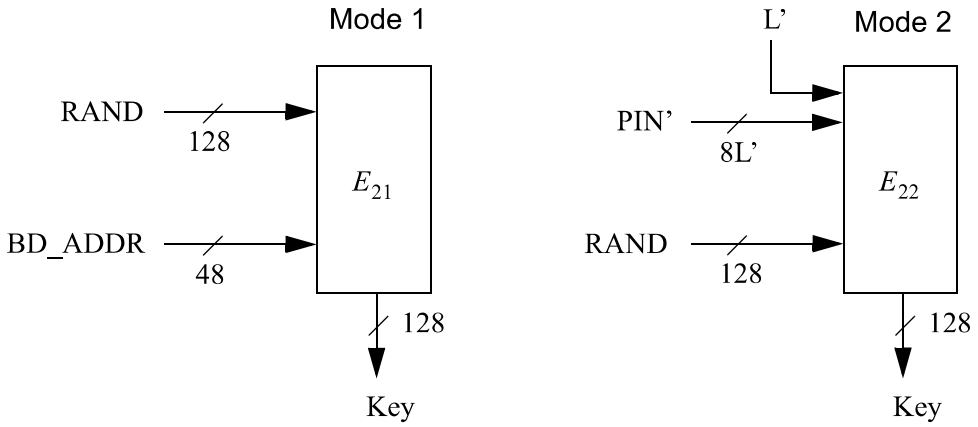


Figure 87—Key generating algorithm E_2 and its two modes

This key-generating algorithm again exploits the cryptographic function. Formally E_2 can be expressed for mode 1 (denoted E_{21}) as

$$E_{21}: \{0,1\}^{128} \times \{0,1\}^{48} \rightarrow \{0,1\}^{128}$$
$$(\text{RAND}, \text{address}) \mapsto A'_r(X, Y) \tag{40}$$

where (for mode 1)

$$\begin{cases} X = \text{RAND}[0 \ldots 14] \cup (\text{RAND}[15] \oplus 6) \\ Y = \bigcup_{i=0}^{15} \text{address}[i \pmod 6] \end{cases} \tag{41}$$

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

$$\text{PIN}' = \begin{cases} \text{PIN}[0 \ldots L-1] \cup \text{BD\_ADDR}[0 \ldots \min\{5, 15-L\}], & L < 16, \\ \text{PIN}[0 \ldots L-1], & L = 16, \end{cases} \tag{42}$$

Then, in mode 2, E_2 (denoted E_{22}) can be expressed as

$$E_{22}: \{0,1\}^{8L'} \times \{0,1\}^{128} \times \{1, 2, \ldots, 16\} \rightarrow \{0,1\}^{128}$$
$$(\text{PIN}', \text{RAND}, \text{L}') \mapsto A'_r(X, Y) \tag{43}$$

Copyright © 2002 IEEE. All rights reserved.

LINKPLAY EXHIBIT 1008
Page 168 of 1169

where

$$\begin{cases} X = \bigcup_{i=0}^{15} \text{PIN}'[i \pmod{L'}], \\ Y = \text{RAND}[0 \ldots 14] \cup (\text{RAND}[15] \oplus L'), \end{cases} \tag{44}$$

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

8.14.5.4 E_3-Key generation function for encryption

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

$$\begin{aligned} E_3\colon \{0,1\}^{128} \times \{0,1\}^{128} \times \{0,1\}^{96} &\rightarrow \{0,1\}^{128} \\ (K, \text{RAND}, \text{COF}) &\mapsto Hash(K, \text{RAND}, \text{COF}, 12) \end{aligned} \tag{45}$$

where $Hash$ is the hash function as defined by Equation (35). Note that the produced key length is 128 bits. However, before use within E_0, the encryption key K_C will be shortened to the correct encryption key length, as described in 8.14.3.5. A block scheme of E_3 is depicted in Figure 88.

The value of COF is determined as specified by equation Equation (25).

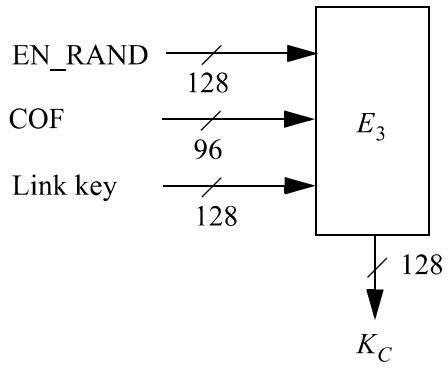


Figure 88—Generation of the encryption key

Copyright © 2002 IEEE. All rights reserved.

9. Link Manager Protocol

Figure 89 indicates the relationship of the Bluetooth protocol stack to this clause. This clause describes the Link Manager Protocol (LMP) which is used for link setup and control. The signals are interpreted and filtered out by the Link Manager on the receiving side and are not propagated to higher layers.

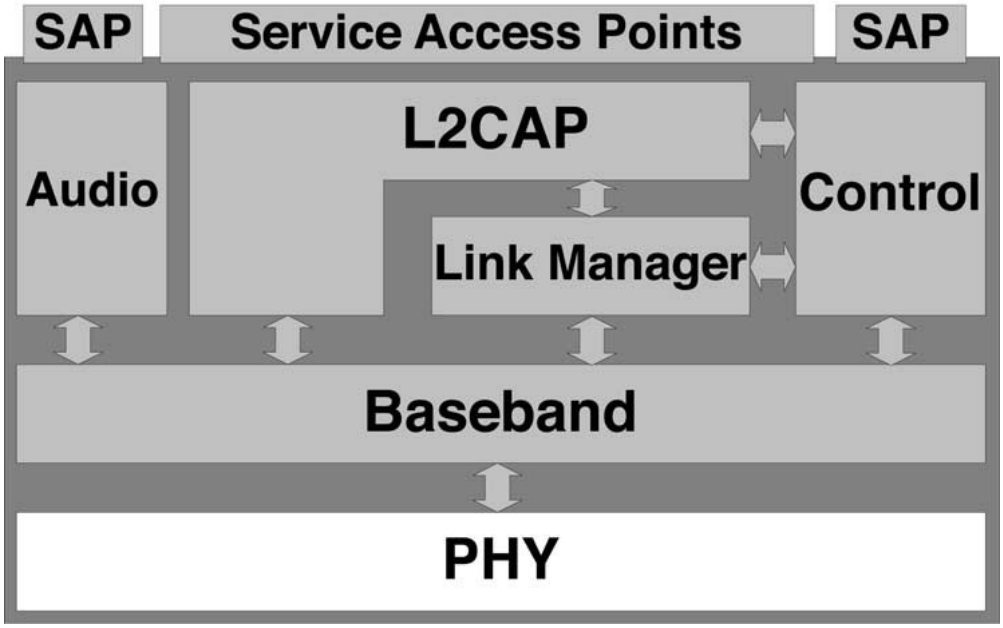


Figure 89—LM interface relationships

9.1 General

LMP messages are used for link setup, security, and control. They are transferred in the payload instead of L2CAP and are distinguished by a reserved value in the L_CH field of the payload header. The messages are filtered out and interpreted by LM on the receiving side and are not propagated to higher layers (see Figure 90).

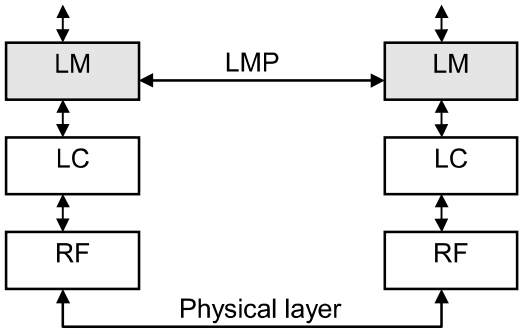


Figure 90—Link Manager's place on the global scene

Copyright © 2002 IEEE. All rights reserved.

Link Manager messages have higher priority than user data. This means that if the Link Manager needs to send a message, it shall not be delayed by the L2CAP traffic, although it can be delayed by many retransmissions of individual baseband packets.

We do not need to explicitly acknowledge the messages in LMP since LC (see 8.5) provides us with a reliable link.

LC does not guarantee either the time taken to deliver a message to the remote device or the delay between the delivery of the message to the remote device and the reception of the corresponding ACK by the sender. This means that we must be aware of the underlying LC mechanism's limitations to synchronize state changes between master and slave. The criteria for determining when the master can reuse an AM_ADDR following the detach or park of a slave is based on the reception of the Baseband-level acknowledgement. Synchronization of a master-slave switch or the starting of hold mode utilizes the Bluetooth master clock, which the LM reads from the LC.

LC only guarantees that it will attempt to communicate with each slave once per T_{poll} slots.

T_{poll} is the poll interval as described in 9.3.20.

The time between receiving a baseband packet carrying an LMP PDU and sending a baseband packet carrying a valid response PDU, according to the procedure rules in 9.3, shall be less than the LMP Response Timeout. The value of this timeout is 30 s. Note that the LMP Response Timeout is applied not only to sequences described in 9.3, but also to the series of the sequences defined as the transactions in 9.3. It is also applied to the series of the transactions, as long as no L2CAP PDUs are allowed, for example, any transactions until the PDUs LMP_setup_complete are exchanged.

9.2 Format of LMP

LM PDUs are always sent as single-slot packets, and the payload header is, therefore, 1 byte. The two least significant bits in the payload header determine the logical channel. For LM PDUs these bits are set. See Table 41.

Table 41—Logical channel L_CH field contents

L_CH code	Logical channel	Information
00	NA	Undefined
01	UA/I	Continuing L2CAP message
10	UA/I	Start L2CAP message
11	LM	LMP message

The FLOW bit in the payload header is always one and is ignored on the receiving side. Each PDU is assigned a 7-bit opcode used to uniquely identify different types of PDUs (see Table 67). The opcode and a 1-bit transaction ID are positioned in the first byte of the payload body (see Figure 91). The transaction ID is positioned in the LSB. It is 0 if the PDU belongs to a transaction initiated by the master and 1 if the PDU belongs to a transaction initiated by the slave. If the PDU contains one or more parameters these are placed in the payload starting at the second byte of the payload body. The number of bytes used depends on the length of the parameters. If an SCO link is present using HV1 packets and length of *content* is less than

Copyright © 2002 IEEE. All rights reserved.

9 bytes, the PDUs can be transmitted in DV packets. Otherwise DM1 packets must be used. All parameters have little endian format, i.e. the least significant byte is transmitted first.

The source/destination of the PDUs is determined by the AM_ADDR in the packet header.

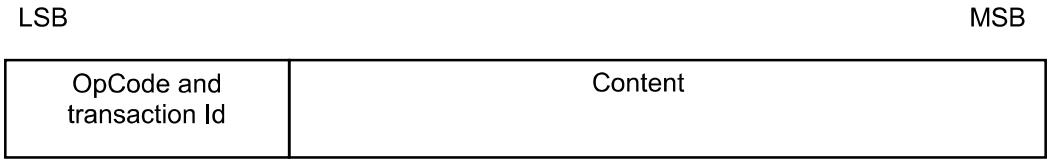


Figure 91—Payload body when LM PDUs are sent

Each PDU is either mandatory or optional. The M/O field in the tables of 9.3 indicates this. The LM does not need to be able to transmit a PDU that is optional. The LM shall recognize all optional PDUs that it receives and, if a response is required, send a valid response according to the procedure rules in 9.3. The reason that should be used in this case is *unsupported LMP feature*. If the optional PDU that is received does not require a response, no response is sent. Which of the optional PDUs a device supports can be requested (see 9.3.11).

Each sequence described in 9.3 is normally defined as a transaction. For pairing (see 9.3.3) or encryption (see 9.3.6), all sequences belonging to each section are counted as one transaction and shall use the same transaction ID. For connection establishment (see 9.4), LMP_host_connection_req and the response with LMP_accepted or LMP_not_accepted form one transaction and have the transaction ID of 0. LMP_setup_complete is a stand-alone PDU, which forms a transaction by itself. For error handling (see 9.7), the PDU to be rejected and LMP_not_accepted form a single transaction. Therefore the LMP_not_accepted shall have the same transaction ID as the PDU which is being rejected.

9.3 The Procedure rules and PDUs

Each procedure is described and depicted with a sequence diagram (see also Figure 92). The following symbols are used in the sequence diagrams:

PDU1 is a PDU sent from A to B. PDU2 is a PDU sent from B to A. PDU3 is a PDU that is optionally sent from A to B. PDU4 is a PDU that is optionally sent from B to A. PDU5 is a PDU sent from either A or B. A vertical line indicates that more PDUs can optionally be sent.

Copyright © 2002 IEEE. All rights reserved.

LINKPLAY EXHIBIT 1008
Page 172 of 1169

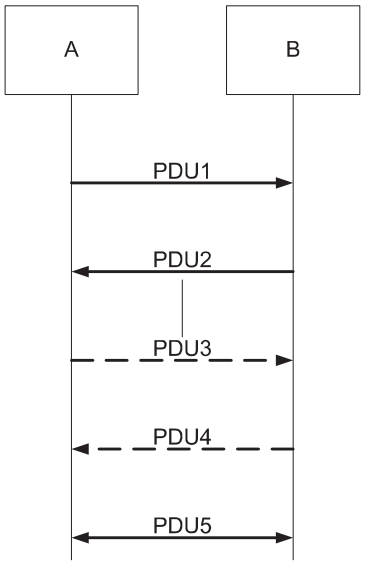


Figure 92—Symbols used in sequence diagrams

9.3.1 General response messages

The PDUs LMP_accepted and LMP_not_accepted are used as response messages to other PDUs in a number of different procedures. The PDU LMP_accepted includes the opcode of the message that is accepted. The PDU LMP_not_accepted includes the opcode of the message that is not accepted and the reason why it is not accepted. See Table 42.

Table 42—General response messages

M/O	PDU	Contents
M	LMP_accepted	op code
M	LMP_not_accepted	op code reason

9.3.2 Authentication

The authentication procedure is based on a challenge-response scheme as described in 8.14.4. The verifier sends an LMP_au_rand PDU which contains a random number (the challenge) to the claimant. The claimant calculates a response, which is a function of the challenge, the claimant's BD_ADDR and a secret key. The response is sent back to the verifier, which checks if the response was correct or not. How the response should be calculated is described in 8.14.5.1. A successful calculation of the authentication response requires that two devices share a secret key. How this key is created is described in 9.3.3. Both the master and the slave can be verifiers. The following PDUs, summarized in Table 43, are used in the authentication procedure:

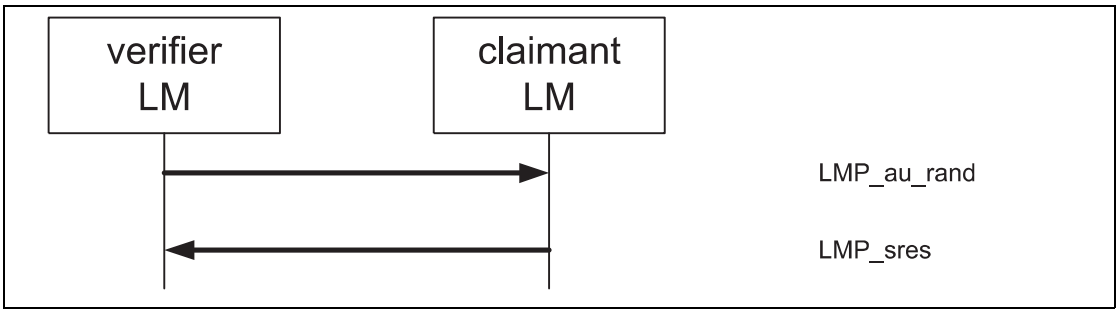
Copyright © 2002 IEEE. All rights reserved.

Table 43—PDUs used for authentication

M/O	PDU	Contents
M	LMP_au_rand	random number
M	LMP_sres	authentication response

9.3.2.1 Claimant has link key

If the claimant has a link key associated with the verifier, it calculates the response and sends it to the verifier with LMP_sres. The verifier checks the response. If the response is not correct, the verifier can end the connection by sending LMP_detach with the reason code *authentication failure (*see 9.3.14).

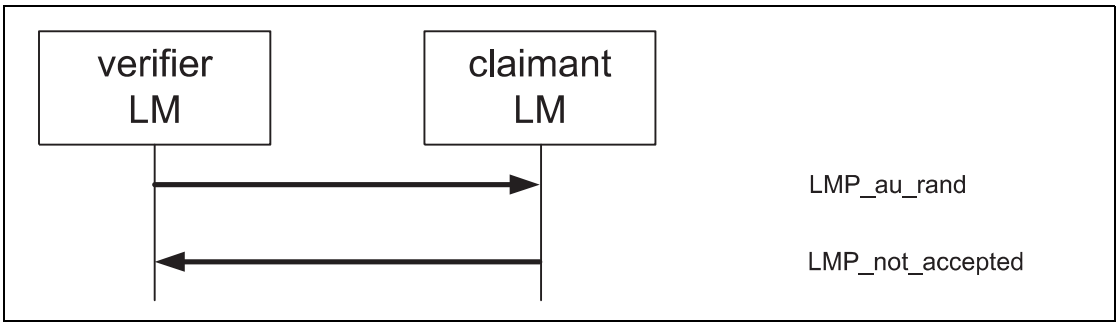


Sequence 1: Authentication. Claimant has link key.

If an LM receives LMP_au_rand and also wants to initiate an authentication it shall first reply with LMP_sres before starting its own challenge. There can, however, be concurrent requests caused by master and slave simultaneously initiating an authentication. To avoid that this results in different ACOs in the units, this situation is resolved by the rule outlined in 9.7: If the master sends LMP_au_rand and receives another LMP_au_rand before receiving LMP_sres it shall respond with LMP_not_accepted with the reason code *LMP Error Transaction Collision;* in that case the slave LM shall complete the master's challenge by sending LMP_sres and may then initiate its authentication again.

9.3.2.2 Claimant has no link key

If the claimant does not have a link key associated with the verifier it sends LMP_not_accepted with the reason code *key missing* after receiving LMP_au_rand.



Sequence 2: Authentication fails. Claimant has no link key.

Copyright © 2002 IEEE. All rights reserved.

LINKPLAY EXHIBIT 1008
Page 174 of 1169

9.3.2.3 Repeated attempts

The scheme described in 8.14.4.1 shall be applied when an authentication fails. This will prevent an intruder from trying a large number of keys in a relatively short time.

9.3.3 Pairing

When two devices do not have a common link key an initialization key (K_{init}) is created based on a PIN and a random number and a BD address. How the K_{init} is calculated is described in 8.14.5.3. When both devices have calculated K_{init} the link key is created, and finally a mutual authentication is made. The pairing procedure starts with a device sending LMP_in_rand; this device is referred to as "initiating LM" or "initiator" in 9.3.3.1 through 9.3.3.5. The other device is referred to as "responding LM" or "responder". The PDUs used in the pairing procedure are summarized in Table 44.

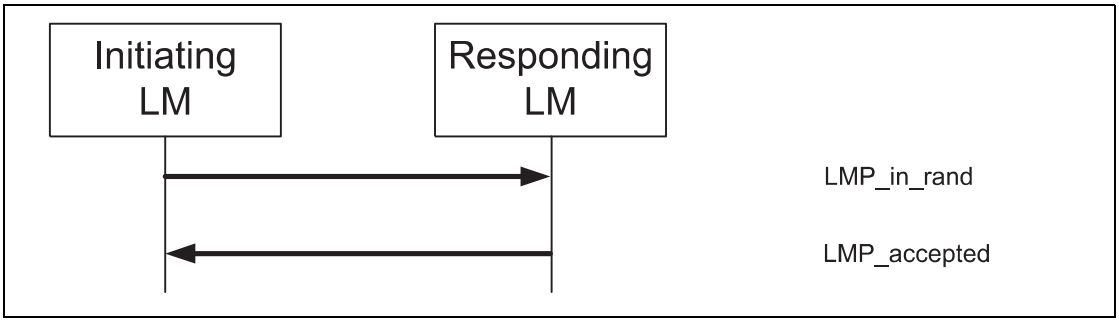
Table 44—PDUs used for pairing

M/O	PDU	Contents
M	LMP_in_rand	random number
M	LMP_au_rand	random number
M	LMP_sres	authentication response
M	LMP_comb_key	random number
M	LMP_unit_key	key

Note that all sequences described in 9.3, including the mutual authentication after the link key has been created, form a single transaction. The transaction ID from the first LMP_in_rand will be used for all subsequent sequences.

9.3.3.1 Responder accepts pairing

The initiator sends LMP_in_rand and the responder replies with LMP_accepted. Both devices calculate K_{init} based on the BD address of the responder and the procedure continues with creation of the link key (see 9.3.3.4).

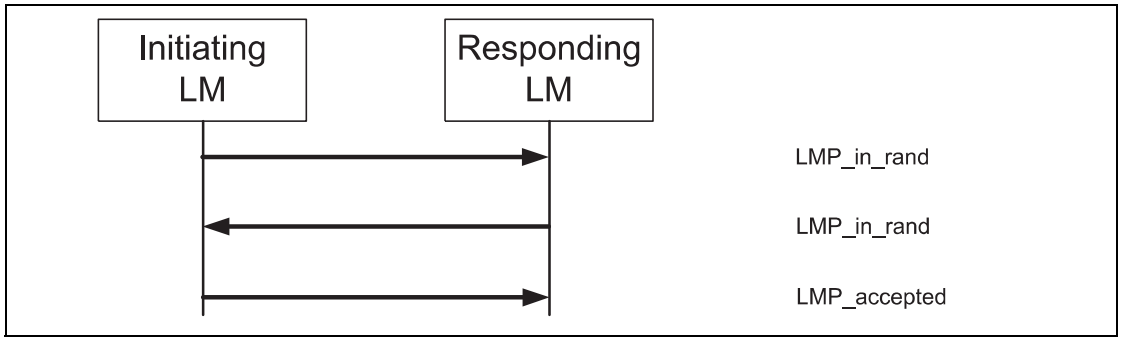


Sequence 3: Pairing accepted. Responder has a variable PIN. Initiator has a variable or fixed PIN.

Copyright © 2002 IEEE. All rights reserved.

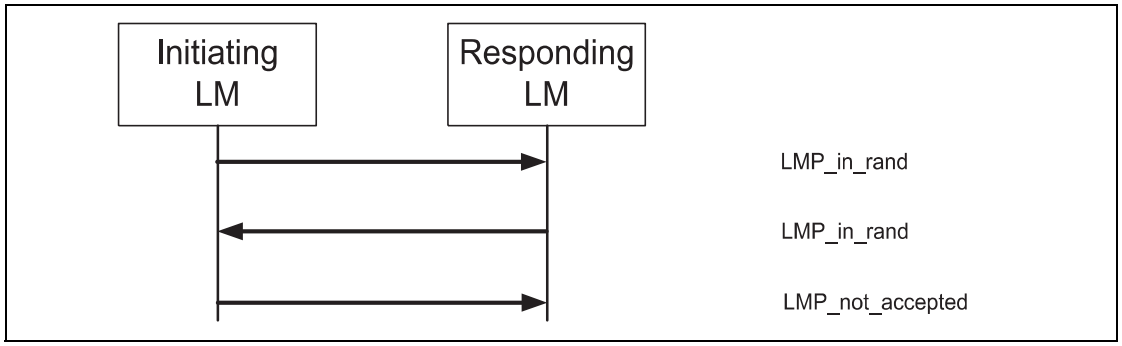
9.3.3.2 Responder has a fixed PIN

If the responder has a fixed PIN, it shall generate a new random number and send it back in LMP_in_rand. If the initiator has a variable PIN, it shall accept this and respond with LMP_accepted. Both sides then calculate K_{init} based on the last IN_RAND and the BD address of the initiator. Thereafter the procedure continues with creation of the link key (see 9.3.3.4).



Sequence 4: Responder has a fixed PIN and initiator has a variable PIN.

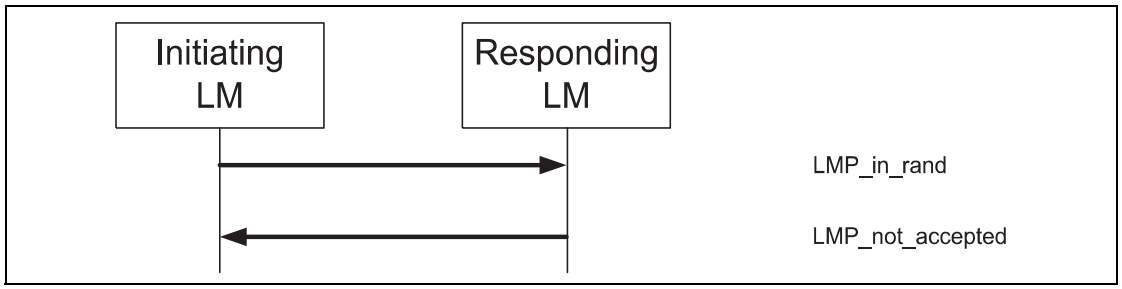
If the responder has a fixed PIN and the initiator also has a fixed PIN, the second LMP_in_rand is rejected by the initiator sending LMP_not_accepted with the reason code *pairing not allowed*; the pairing procedure is then ended.



Sequence 5: Both devices have a fixed PIN.

9.3.3.3 Responder rejects pairing

If the responder rejects pairing it sends LMP_not_accepted with the reason code *pairing not allowed* after receiving LMP_in_rand.



Sequence 6: Responder rejects pairing.

Copyright © 2002 IEEE. All rights reserved.

9.3.3.4 Creation of the link key

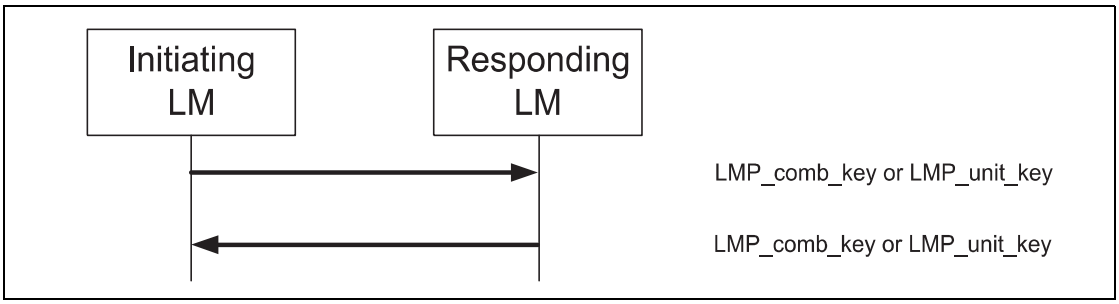
When K_{init} is calculated in both units the link key must be created. This link key will be used in the authentication between the two units for all subsequent connections until it is changed (see 9.3.4 and 9.3.5). The link key created in the pairing procedure will either be a combination key or one of the unit's unit keys. The following rules apply to the selection of the link key:

— If one unit sends LMP_unit_key and the other unit sends LMP_comb_key, the unit key will be the link key.

— If both units send LMP_unit_key, the master's unit key will be the link key.

— If both units send LMP_comb_key, the link key is calculated as described in 8.14.2.2.

The content of LMP_unit_key is the unit key bitwise XORed with K_{init}. The content of LMP_comb_key is LK_RAND bitwise XORed with K_{init}. Any device configured to use a combination key will store the link key.



Sequence 7: Creation of the link key.

9.3.3.5 Repeated attempts

When the authentication after creation of the link key fails because of a wrong authentication response, the same scheme as in 9.3.2.3 is applied. This prevents an intruder from trying a large number of different PINs in a relatively short time.

9.3.4 Change link key

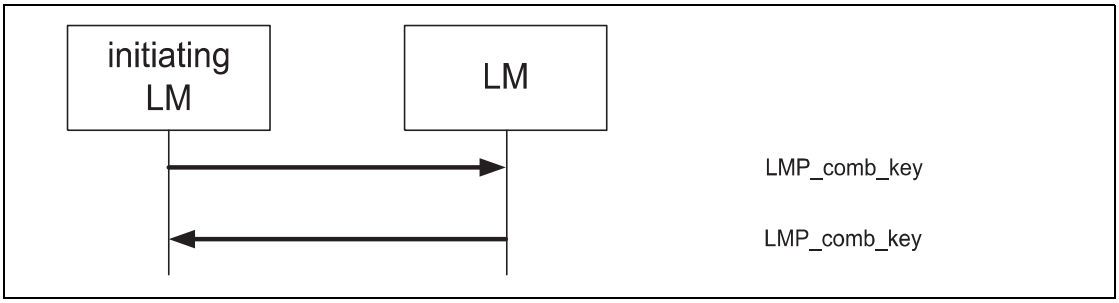
If the link key is derived from combination keys and the current link is the semipermanent link key, the link key can be changed. If the link key is a unit key, the units shall go through the pairing procedure in order to change the link key. The contents of LMP_comb_key is protected by a bitwise XOR with the current link key and summarized in Table 45.

Table 45—PDUs used for change of link key

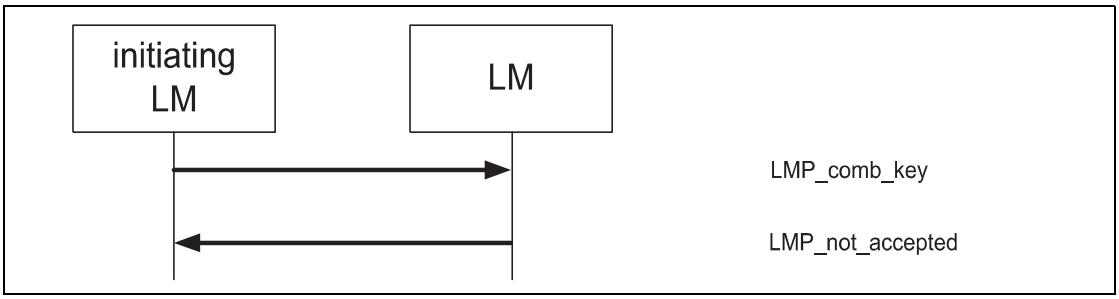
M/O	PDU	Contents
M	LMP_comb_key	random number

Copyright © 2002 IEEE. All rights reserved.

Note that all sequences described in 9.3.4, including the mutual authentication after the link key has been changed, form a single transaction. The transaction ID from the first LMP_comb_key will be used for all subsequent sequences.



Sequence 8: Successful change of the link key.



Sequence 9: Change of the link key not possible since the other unit uses a unit key.

If the change of link key is successful the new link key is stored and the old link key is discarded. The new link key will be used as link key for all the following connections between the two devices until the link key is changed again. The new link key also becomes the current link key. It will remain the current link key until the link key is changed again, or until a temporary link key is created (see 9.3.5).

When the new link key has been created mutual authentication shall be made to confirm that the same link key has been created in both units. The first authentication in the mutual authentication is made with the unit that initiated change link key as verifier. When finalized an authentication in the reversed direction is made.

9.3.5 Change the current link key

The current link key can be a semipermanent link key or a temporary link key key. It can be changed temporarily, but the change is only valid for the session (see 8.14.2.1). Changing to a temporary link key is necessary if the piconet is to support encrypted broadcast (see Table 46).

9.3.5.1 Change to a temporary link key

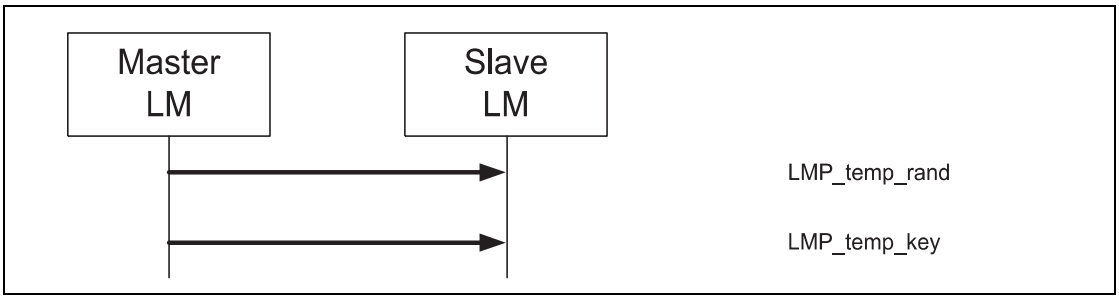
In the following, the same terms as in 8.14.2.2.8. The master starts by creating the master key K_{master} as described in Equation (26). Then the master issues a random number RAND and sends it to the slave in LMP_temp_rand. Both sides can then calculate an overlay denoted OVL as OVL= E_{22}(current link key, RAND, 16). Then the master sends K_{master} protected by a modulo-2 addition with OVL to the slave in LMP_temp_key. The slave, who knows OVL, calculates K_{master}. After this, K_{master} becomes the current link key. It will be the current link key until the units fall back to the semipermanent link key (see 9.3.5.2).

Copyright © 2002 IEEE. All rights reserved.

LINKPLAY EXHIBIT 1008
Page 178 of 1169

Table 46—PDUs used to change the current link key

M/O	PDU	Contents
M	LMP_temp_rand	random number
M	LMP_temp_key	key
M	LMP_use_semi_permanent_key	—



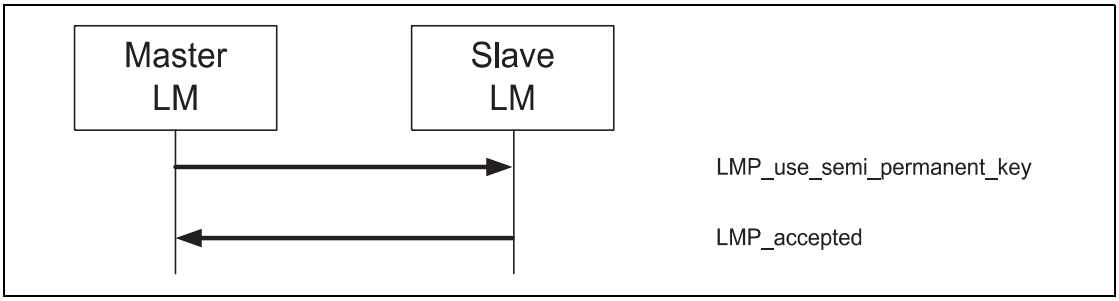
Sequence 10: Change to a temporary link key.

Note that all sequences described in 9.3.4, including the mutual authentication after K_{master} has been created, form a single transaction. The transaction ID is set to 0.

When the units have changed to the temporary key, a mutual authentication shall be made to confirm that the same link key has been created in both units. The first authentication in the mutual authentication is made with the master as verifier. When finalized an authentication in the reversed direction is made.

9.3.5.2 Make the semipermanent link key the current link key

After the current link key has been changed to K_{master}, this change can be undone and the semipermanent link key becomes the current link key again. If encryption is used on the link, the procedure of going back to the semipermanent link key shall be immediately followed by a stop of the encryption by the master invoking the procedure described in 9.3.6.4. Encryption can then be started again by the master according to the procedures in 9.3.6.1. This is to assure that encryption with encryption parameters known by other devices in the piconet is not used when the semipermanent link key is the current link key.



Sequence 11: Link key changed to the semipermanent link key.

Copyright © 2002 IEEE. All rights reserved.

9.3.6 Encryption

If at least one authentication has been performed encryption may be used. If the master wants all slaves in the piconet to use the same encryption parameters it must issue a temporary key (K_{master}) and make this key the current link key for all slaves in the piconet before encryption is started (see 9.3.5). This is necessary if broadcast packets should be encrypted (see Table 47).

Table 47—PDUs used for handling encryption

M/O	PDU	Contents
O	LMP_encryption_mode_req	encryption mode
O	LMP_encryption_key_size_req	key size
O	LMP_start_encryption_req	random number
O	LMP_stop_encryption_req	—

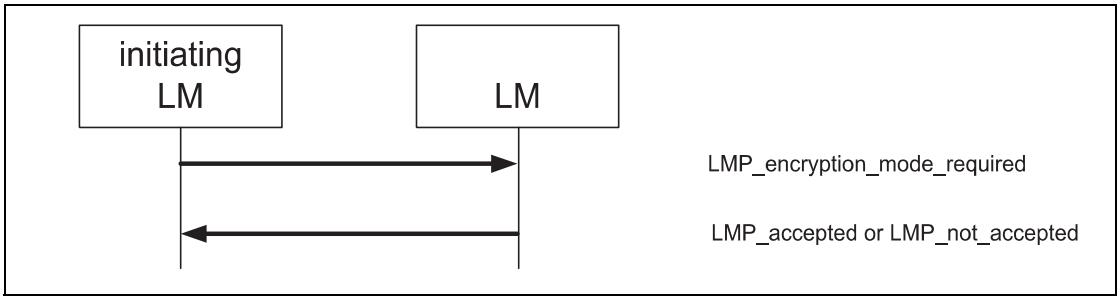
Note that all sequences described in 9.3.6 form a single transaction. The transaction ID from the LMP_encryption_mode_req will be used for all subsequent sequences.

9.3.6.1 Encryption mode

First of all the master and the slave must agree upon whether to use encryption or not and if encryption shall only apply to point-to-point packets or if encryption shall apply to both point-to-point packets and broadcast packets. If master and slave agree on the encryption mode, the master continues to give more detailed information about the encryption.

The initiating LM finalizes the transmission of the current ACL packet with L2CAP information, stops L2CAP transmission and sends LMP_encryption_mode_req. If the change in encryption mode is accepted then the other device finalizes the transmission of the current ACL packet with L2CAP information, stops L2CAP transmission and responds with LMP_accepted.

L2CAP transmission is re-enabled when the attempt to encrypt or decrypt the link is completed i.e., at the end of Sequence 14, 15, or 16.



Sequence 12: Negotiation for encryption mode.

After a unit has sent LMP_encryption_mode_req, it is not allowed to send LMP_au_rand before encryption is actually switched on. After a unit has received LMP_encryption_mode_req and sent LMP_accepted it is not allowed to send LMP_au_rand before encryption is actually switched on. If an LMP_au_rand is still sent

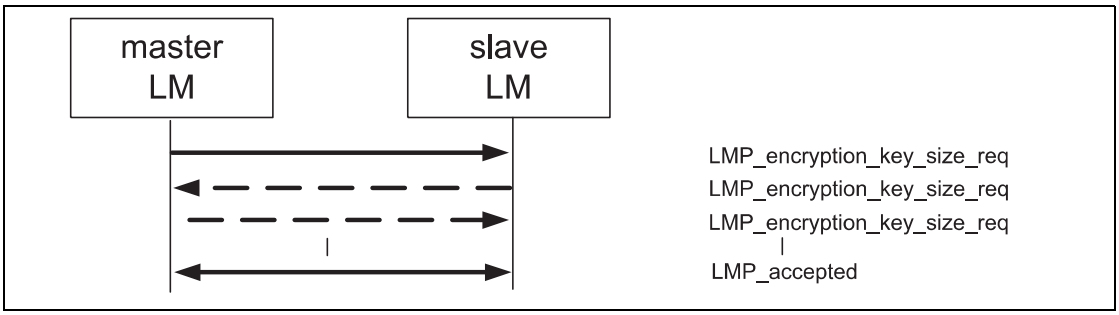
Copyright © 2002 IEEE. All rights reserved.

LINKPLAY EXHIBIT 1008
Page 180 of 1169

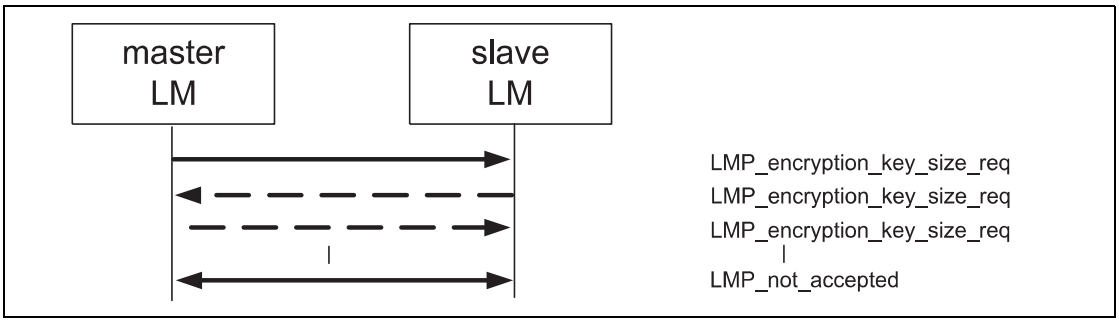
violating these rules, the claimant shall respond with LMP_not_accepted with the reason code *PDU not allowed*. This is to avoid that the units have different ACOs when they calculate the encryption key. If the encryption mode is not accepted or the encryption key size negotiation results in disagreement the units are allowed to send LMP_au_rand again.

9.3.6.2 Encryption key size

The next step is to determine the size of the encryption key. In the following we use the same terms as in 8.14.3.1. The master sends LMP_encryption_key_size_req including the suggested key size $L_{sug,\ m}$, which is initially equal to $L_{max,\ m}$. If $L_{min,\ s} \leq L_{sug,\ m}$ and the slave supports $L_{sug,\ m}$ it responds with LMP_accepted and $L_{sug,\ m}$ will be used as the key size. If both conditions are not fulfilled the slave sends back LMP_encryption_key_size_req including the slave's suggested key size $L_{sug,\ s}$. This value is the slave's largest supported key size that is less than $L_{sug,\ m}$. Then the master performs the corresponding test on the slave's suggestion. This procedure is repeated until a key size agreement is reached or it becomes clear that no such agreement can be reached. If an agreement is reached a unit sends LMP_accepted and the key size in the last LMP_encryption_key_size_req will be used. After this, the encryption is started; see 9.3.6.3. If an agreement is not reached a unit sends LMP_not_accepted with the reason code *Unsupported parameter value* and the units are not allowed to communicate using Bluetooth link encryption.



Sequence 13: Encryption key size negotiation successful.

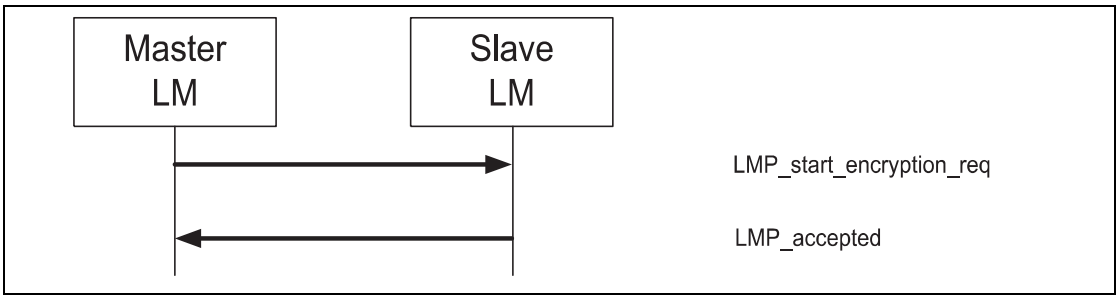


Sequence 14: Encryption key size negotiation failed.

9.3.6.3 Start encryption

Finally, encryption is started. The master issues the random number EN_RAND and calculates the encryption key as $K_c=E_3$(current link key, EN_RAND, COF). See 8.14.2.2.5 for the definition of the COF. The random number must be the same for all slaves if the piconet should support encrypted broadcast. Then the master sends LMP_start_encryption_req, which includes EN_RAND. The slave calculates K_c when this message is received and acknowledges with LMP_accepted.

Copyright © 2002 IEEE. All rights reserved.



Sequence 15: Start of encryption.

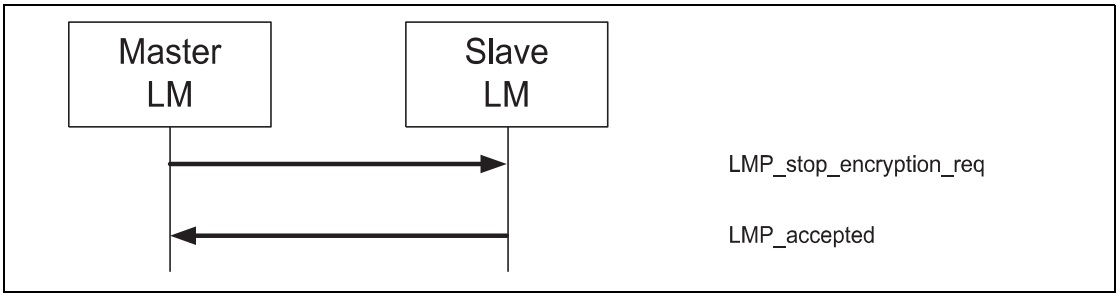
The start of encryption will be done in three steps:

1) Master is configured to transmit unencrypted packets, but to receive encrypted packets.
2) Slave is configured to transmit and receive encrypted packets.
3) Master is configured to transmit and receive encrypted packets.

Between step 1 and step 2, master-to-slave transmission is possible. This is when LMP_start_encryption_req is transmitted. Step 2 is triggered when the slave receives this message. Between step 2 and step 3, slave-to-master transmission is possible. This is when LMP_accepted is transmitted. Step 3 is triggered when the master receives this message.

9.3.6.4 Stop encryption

If a unit wants to stop encryption it sends LMP_encryption_mode_req with the parameter encryption mode equal to 0 (no encryption). The other device responds with LMP_accepted or LMP_not_accepted (the procedure is described in Sequence 12 in 9.3.6.1). If accepted encryption is stopped by the master sending LMP_stop_encryption_req and the slave responding with LMP_accepted according to Sequence 16.



Sequence 16: Stop of encryption.

Stopping of encryption is then done in three steps, similar to the procedure for starting encryption.

1) Master is configured to transmit encrypted packets, but to receive unencrypted packets.
2) Slave is configured to transmit and receive unencrypted packets.
3) Master is configured to transmit and receive unencrypted packets.

Between step 1 and step 2 master to slave transmission is possible. This is when LMP_stop_encryption_req is transmitted. Step 2 is triggered when the slave receives this message. Between step 2 and step 3 slave to

Copyright © 2002 IEEE. All rights reserved.

LINKPLAY EXHIBIT 1008
Page 182 of 1169

master transmission is possible. This is when LMP_accepted is transmitted. Step 3 is triggered when the master receives this message.

9.3.6.5 Change encryption mode, key or random number

If the encryption key or encryption random number need to be changed or if the encryption mode needs to be changed between 1 (point-to-point) and 2 (point-to-point and broadcast), encryption must first be stopped and then restarted with the new parameters according to the procedures in 9.3.6.

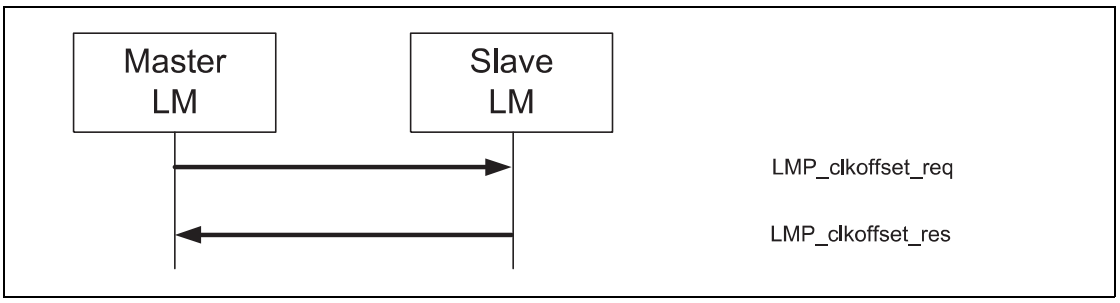
9.3.7 Clock offset request

When a slave receives the FHS packet, the difference is computed between its own clock and the master's clock included in the payload of the FHS packet. The clock offset is also updated each time a packet is received from the master. The master can request the clock offset at anytime following a successful baseband paging procedure (i.e., before, during, or after connection setup).

By saving this clock offset, the master knows on what RF channel the slave wakes up to PAGE SCAN after it has left the piconet. This can be used to speed up the paging time the next time the same device is paged (see Table 48).

Table 48—PDUs used for clock offset request

M/O	PDU	Contents
M	LMP_clkoffset_req	—
M	LMP_clkoffset_res	clock offset



Sequence 17: Clock offset requested.

9.3.8 Slot offset information

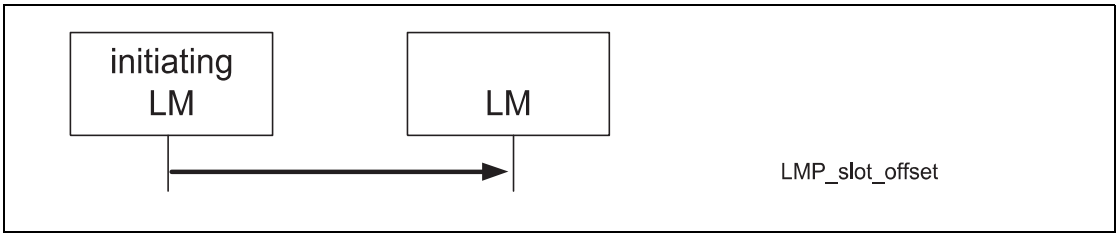
With LMP_slot_offset the information about the difference between the slot boundaries in different piconets is transmitted. This PDU carries the parameters slot offset and BD_ADDR. The slot offset is the subtraction of time in µs of the start of the master's TX slot in the piconet where the PDU is transmitted from the time in µs of the start of the master's TX slot in the piconet where the BD_ADDR device is master modulo 1250 (see Table 49).

See 9.3.12 for the use of LMP_slot_offset in the context of the master-slave switch.

Copyright © 2002 IEEE. All rights reserved.

Table 49— PDU used for slot offset information

M/O	PDU	Contents
O	LMP_slot_offset	slot offset BD_ADDR



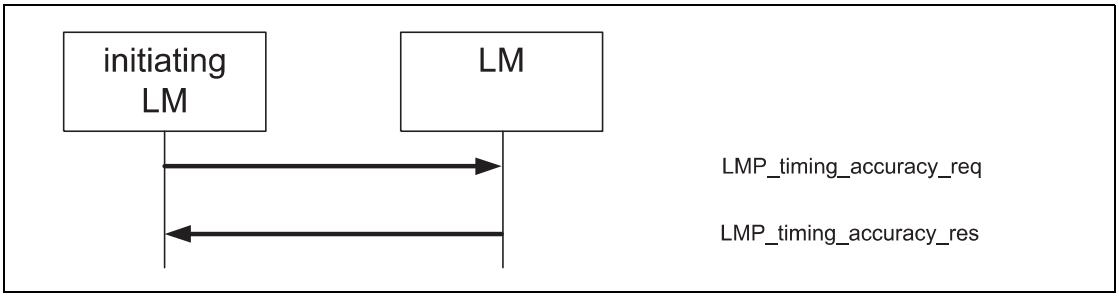
Sequence 18: Slot offset information is sent.

9.3.9 Timing accuracy information request

LMP supports requests for the timing accuracy. This information can be used to minimize the scan window for a given hold time when returning from hold and to extend the maximum hold time. It can also be used to minimize the scan window when scanning for the sniff mode slots or the park mode beacon packets. The timing accuracy parameters returned are the long-term drift measured in ppm and the long term jitter measured in µs of the clock used during hold, sniff, and park mode. These parameters are fixed for a certain device and shall be identical when requested several times. Timing accuracy can be requested at anytime following a successful baseband paging procedure, provided this PDU is shown as supported in the supported features list. If the timing accuracy request is not supported, the requesting device shall assume worst case values (drift = 250 ppm and jitter = 10 µs) (see Table 50).

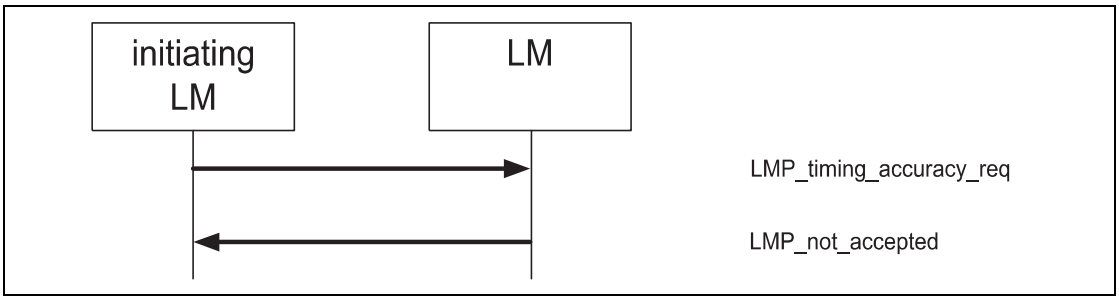
Table 50—PDUs used for requesting timing accuracy information

M/O	PDU	Contents
O	LMP_timing_accuracy_req	—
O	LMP_timing_accuracy_res	drift jitter



Sequence 19: The requested device supports timing accuracy information.

Copyright © 2002 IEEE. All rights reserved.



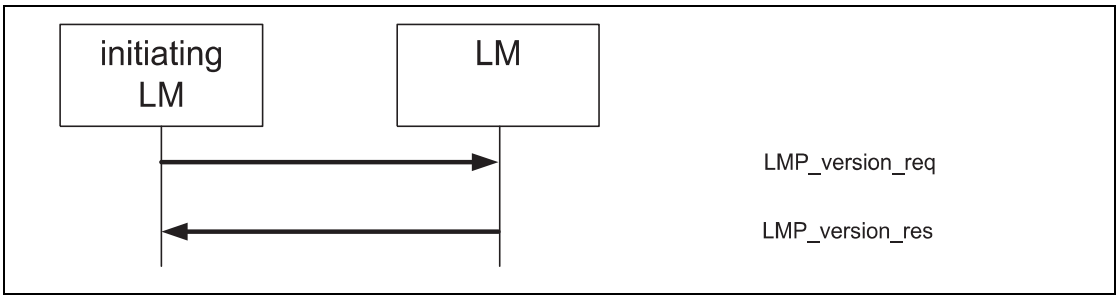
Sequence 20: The requested device does not support timing accuracy information.

9.3.10 LMP version

LMP supports requests for the version of the LM protocol. The requested device will send a response with three parameters: VersNr, CompId, and SubVersNr. VersNr specifies the version of the Bluetooth LMP specification that the device supports. CompId is used to track possible problems with the lower Bluetooth layers. All companies that create a unique implementation of the Link Manager shall have their own CompId. The same company is also responsible for the administration and maintenance of the SubVersNr. It is recommended that each company have a unique SubVersNr for each RF/BB/LM implementation. For a given VersNr and CompId, the values of the SubVersNr shall increase each time a new implementation is released. For both CompId and SubVersNr the value 0xFFFF means that no valid number applies. There is no ability to negotiate the version of the LMP. The sequence below (Sequence 21) is only used to exchange the parameters. LMP version can be requested at anytime following a successful baseband paging procedure (see Table 51).

Table 51—PDUs used for LMP version request

M/O	PDU	Contents
M	LMP_version_req	VersNr CompId SubVersNr
M	LMP_version_res	VersNr CompId SubVersNr



Sequence 21: Request for LMP version.

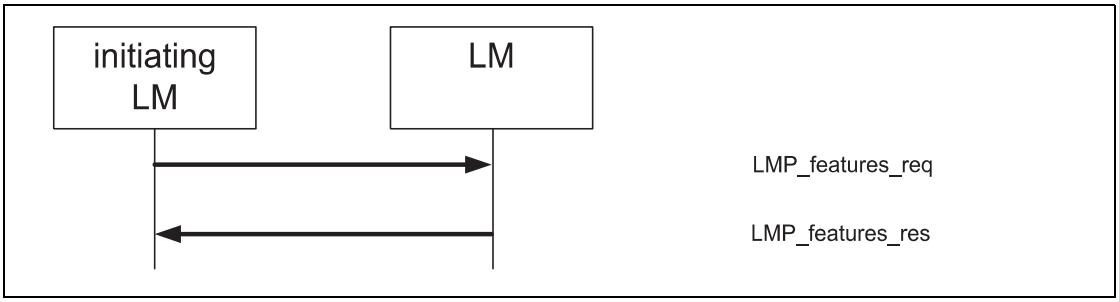
Copyright © 2002 IEEE. All rights reserved.
LINKPLAY EXHIBIT 1008
Page 185 of 1169

9.3.11 Supported features

The Bluetooth radio and link controller may support only a subset of the packet types and features described in Clause 8 and Physical layer. The PDU LMP_features_req and LMP_features_res are used to exchange this information. The supported features can be requested at anytime following a successful baseband paging procedure. A device may not send any packets other than ID, FHS, NULL, POLL, DM1 or DH1 before it is aware of the supported features of the other device. After the features request has been carried out, the intersection of the supported packet types for both sides may also be transmitted. Whenever a request is issued, it shall be compatible with the supported features of the other device. For instance, when establishing an SCO link the initiator may not propose to use HV3 packets if that packet type is not supported by the other device. Exceptions to this rule are LMP_switch_ req and LMP_slot_offset, which can be sent before the requesting side is aware of the other side's features (switch is an optional feature) (see Table 52).

Table 52—PDUs used for features request

M/O	PDU	Contents
M	LMP_features_req	features
M	LMP_features_res	features



Sequence 22: Request for supported features.

9.3.12 Switch of master-slave role

Since the paging device always becomes the master of the piconet, a switch of the master-slave role is sometimes needed (see 8.10.9.3). See also Table 53.

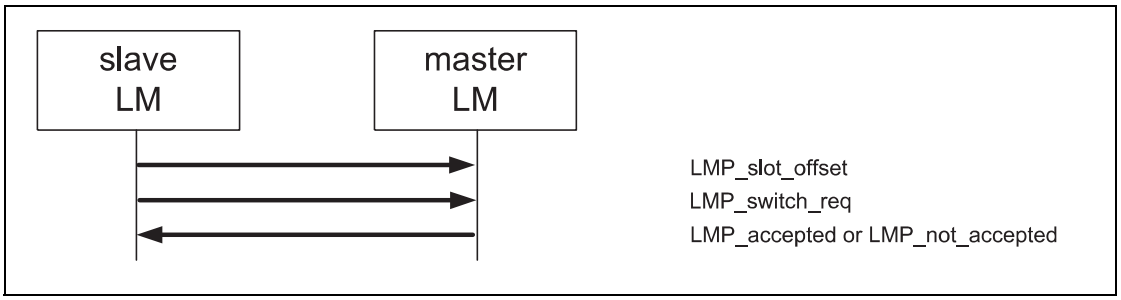
Table 53—PDUs used for master-slave switch

M/O	PDU	Contents
O	LMP_switch_req	switch instant
O	LMP_slot_offset	slot offset BD_ADDR

Copyright © 2002 IEEE. All rights reserved.

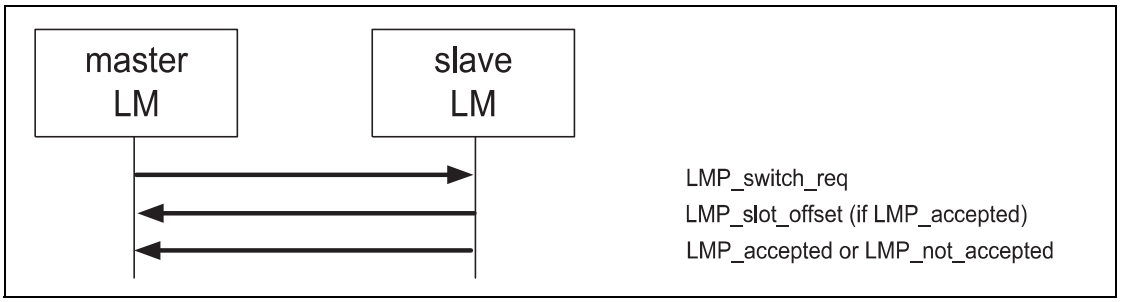
LINKPLAY EXHIBIT 1008
Page 186 of 1169

If the slave initiates the master-slave switch, it finalizes the transmission of the current ACL packet with L2CAP information, stops L2CAP transmission, and sends LMP_slot_offset immediately followed by LMP_switch_req. If the master accepts the master-slave switch, it finalizes the transmission of the current ACL packet with L2CAP information, stops L2CAP transmission, and responds with LMP_accepted. When the master-slave switch has been completed on Baseband level (successfully or not), both units re-enable L2CAP transmission. If the master rejects the master-slave switch, it responds with LMP_not_accepted and the slave re-enables L2CAP transmission. The transaction ID for all PDUs in the sequence is set to 1.



Sequence 23: Master-slave switch (slave initiated).

If the master initiates the master-slave switch, it finalizes the transmission of the current ACL packet with L2CAP information, stops L2CAP transmission, and sends LMP_switch_req. If the slave accepts the master-slave switch, it finalizes the transmission of the current ACL packet with L2CAP information, stops L2CAP transmission, and responds with LMP_slot_offset immediately followed by LMP_accepted. When the master-slave switch has been completed on Baseband level (successfully or not), both units re-enable L2CAP transmission. If the slave rejects the master-slave switch, it responds with LMP_not_accepted and the master re-enables L2CAP transmission. The transaction ID for all PDUs in the sequence is set to 0.



Sequence 24: Master-slave switch (master initiated).

The LMP_switch_req PDU contains a parameter, switch instant, which specifies the instant at which the TDD switch is made. This is specified as a Bluetooth clock value of the master's clock, which is available to both devices. This instant is chosen by the sender of the message and should be at least $2*T_{poll}$ or 32 (whichever is greater) slots in the future. The assumption is that the switch instant is always within 12 h of the current clock value, in order to accommodate clock wrap.

The sender of the LMP_switch_req selects the switch instant, queues the LMP_switch_req to LC for transmission, and starts a timer to expire at the switch instant. When the timer expires it initiates the mode switch. In the case of a master-initiated switch, if the LMP_slot_offset has not been received by the switch instant, the master slave switch is carried out without an estimate of the slave's slot offset. If LMP_not_accepted is received before the timer expires, then the timer is stopped, and the role switch is not initiated.

Copyright © 2002 IEEE. All rights reserved.
LINKPLAY EXHIBIT 1008
Page 187 of 1169

When the LMP_switch_req is received, the switch instant is compared with the current master clock value. If it is in the past, then the instant has been passed and LMP_not_accepted with the reason code Instant passed shall be returned. If it is in the future, then LMP_accepted is returned assuming the master-slave switch is allowed and a timer is started to expire at the switch instant. When this timer expires, it initiates the mode switch.

Support for LMP_slot_offset is mandatory if LMP_switch_req is supported.

9.3.13 Name request

LMP supports name request to another Bluetooth device. The name is a user-friendly name associated with the Bluetooth device and consists of a maximum of 248 bytes coded according to the UTF-8 standard. The name is fragmented over one or more DM1 packets. When the LMP_name_req is sent, a name offset indicates which fragment is expected. The corresponding LMP_name_res carries the same name offset, the name length indicating the total number of bytes in the name of the Bluetooth device and the name fragment, where:

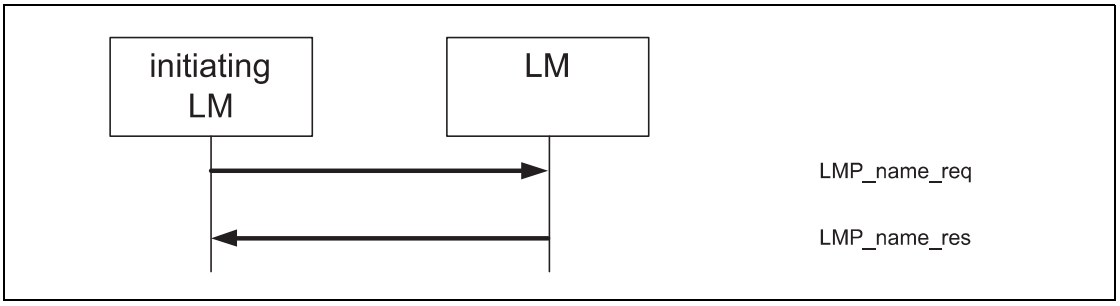
— name fragment(N) = name(N + name offset), if (N + name offset) < name length

— name fragment(N) = 0, otherwise.

Here 0 ≤ N ≤ 13. In the first sent LMP_name_req, name offset = 0. Sequence 25 is then repeated until the initiator has collected all fragments of the name. The name request can be made at anytime following a successful baseband paging procedure (see Table 54).

Table 54—PDUs used for name request

M/O	PDU	Contents
M	LMP_name_req	name offset
M	LMP_name_res	name offset name length name fragment



Sequence 25: Device's name requested and it responses.

Copyright © 2002 IEEE. All rights reserved.

9.3.14 Detach

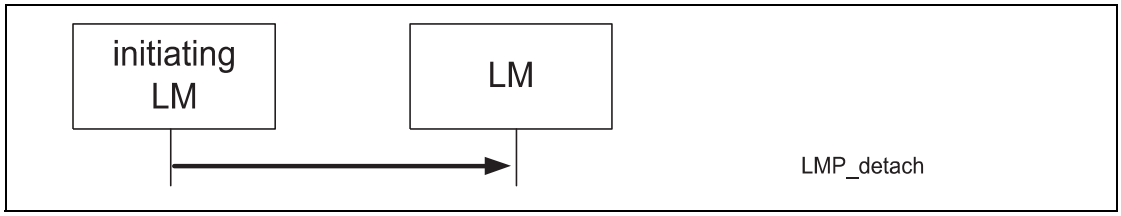
The connection between two Bluetooth devices can be closed anytime by the master or the slave. A reason parameter is included in the message to inform the other party of why the connection is closed (see Table 55).

Table 55—PDU used for detach

M/O	PDU	Contents
M	LMP_detach	reason

The initiating LM first finalizes the transmission of the current ACL packet with L2CAP information and stops L2CAP transmission. The initiating LM then queues the LMP_detach for transmission and starts a timer for $6*T_{poll}$ slots where T_{poll} is the poll interval for the connection. If the initiating LM receives the Baseband-level acknowledgement before the timer expires it now starts a timer for $3*T_{poll}$ slots. When this timer expires (and if the initiating LM is the master) the AM_ADDR can be re-used immediately. If the initial timer expires then the initiating LM drops the link and starts a timer for $T_{linksupervisiontimeout}$ slots after which the AM_ADDR can be re-used (if the initiating LM is the master).

When the receiving LM receives the LMP_detach, it starts a timer for $6*T_{poll}$ slots if it is the master and $3*T_{poll}$ if it is the slave. On timer expiration, the link is detached and (if the receiving LM is the master) the AM_ADDR can be re-used immediately. If the receiver never gets the LMP_detach then a link supervision timeout will occur and the link will be detached.



Sequence 26: Connection closed by sending LMP_detach.

9.3.15 Hold mode

The ACL link of a connection between two Bluetooth devices can be placed in hold mode for a specified hold time. During this time no ACL packets will be transmitted from the master. The hold mode is typically entered when there is no need to send data for a relatively long time. The transceiver can then be turned off in order to save power. But the hold mode can also be used if a device wants to discover or be discovered by other Bluetooth devices, or wants to join other piconets. What a device actually does during the hold time is not controlled by the hold message, but it is up to each device to decide (see Table 56).

The LMP_hold and LMP_hold_req PDUs both contain a parameter, hold instant, which specifies the instant at which the hold will become effective. This is specified as a Bluetooth clock value of the master's clock, which is available to both devices. This instant is chosen by the sender of the message and should be at least $6*T_{poll}$ slots in the future. The assumption is that the hold instant is always within 12 h of the current clock value, in order to accommodate clock wrap.

Copyright © 2002 IEEE. All rights reserved.

Table 56—PDUs used for hold mode

M/O	PDU	Contents
O	LMP_hold	hold time, hold instant
O	LMP_hold_req	hold time, hold instant

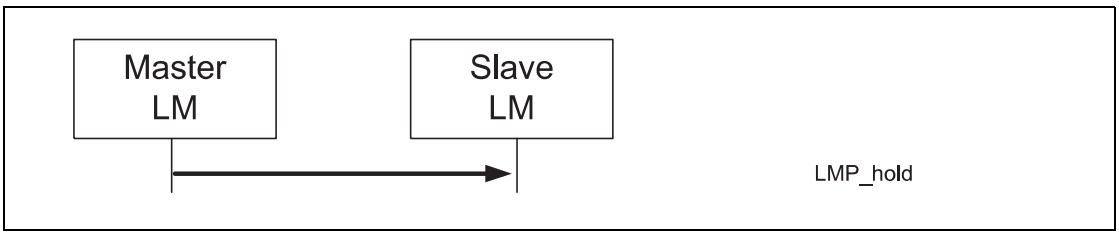
9.3.15.1 Master forces hold mode

The master can force hold mode if there has previously been a request for hold mode that has been accepted. The hold time included in the PDU when the master forces hold mode cannot be longer than any hold time the slave has previously accepted when there was a request for hold mode.

The master LM first finalizes the transmission of the current ACL packet with L2CAP information and stops L2CAP transmission. It selects the hold instant and queues the LMP_hold to its LC for transmission. It then starts a timer to wait until the hold instant occurs. When this timer expires then the connection enters hold mode. Note that the Baseband-level acknowledgement is ignored in this mechanism.

When the slave LM receives the LMP_hold it compares the hold instant with the current master clock value. If it is in the future then it starts a timer to expire at this instant and enters hold mode when it expires.

When the master LM exits from Hold mode it re-enables L2CAP transmission.



Sequence 27: Master forces slave into hold mode.

9.3.15.2 Slave forces hold mode

The slave can force hold mode if there has previously been a request for hold mode that has been accepted. The hold time included in the PDU when the slave forces hold mode cannot be longer than any hold time the master has previously accepted when there was a request for hold mode.

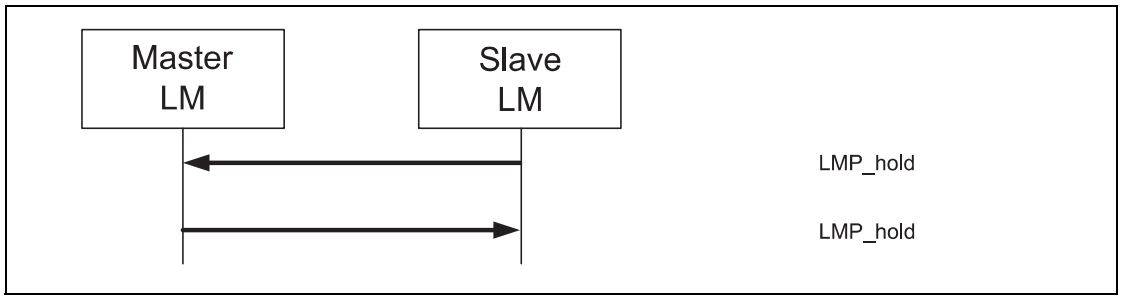
The slave LM first finalizes the transmission of the current ACL packet with L2CAP information and stops L2CAP transmission. It selects the hold instant and queues the LMP_hold to its LC for transmission. It then waits for the LMP_hold from the master acting according to the procedure described in 9.3.15.1.

When the master LM receives the LMP_hold it finalizes the transmission of the current ACL packet with L2CAP information and stops L2CAP transmission. It then inspects the hold instant. If this is less than $6*T_{poll}$ slots in the future it should modify the instant so that it is at least $6*T_{poll}$ slots in the future. Then it sends the LMP_hold using the mechanism described in 9.3.15.1.

When the master and slave LMs exit from Hold mode they re-enable L2CAP transmission.

Copyright © 2002 IEEE. All rights reserved.

LINKPLAY EXHIBIT 1008
Page 190 of 1169



Sequence 28: Slave forces master into hold mode.

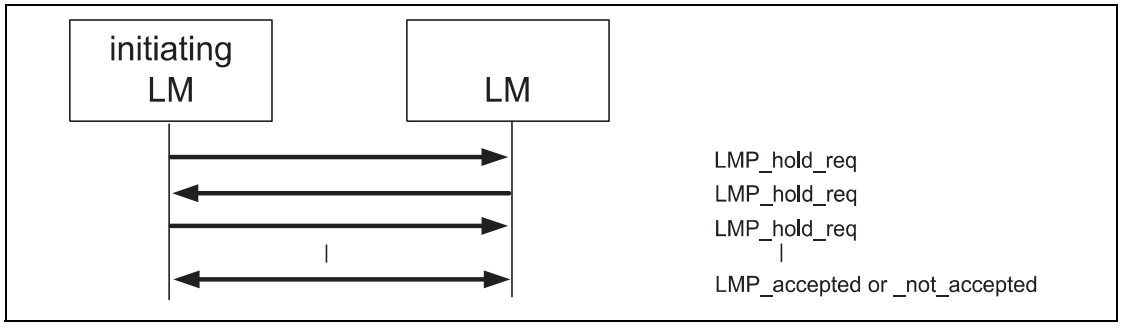
9.3.15.3 Master or slave requests hold mode

The master or the slave can request to enter hold mode. Upon receipt of the request, the same request with modified parameters can be returned or the negotiation can be terminated. If an agreement is seen, LMP_accepted terminates the negotiation and the ACL link is placed in hold mode. If no agreement is seen, LMP_not_accepted with the reason code *unsupported parameter value* terminates the negotiation and hold mode is not entered.

The initiating LM first finalizes the transmission of the current ACL packet with L2CAP information and stops L2CAP transmission. On receiving the LMP_hold_req, the receiving LM finalizes the transmission of the current ACL packet with L2CAP information and stops L2CAP transmission

The LM sending LMP_hold_req selects the hold instant, which should be at least $9*T_{poll}$ slots in the future. If this is a response to a previous LMP_hold_req and the contained hold instant is at least $9*T_{poll}$ slots in the future then this should be used. The LMP_hold_req is then queued to its LC for transmission and a timer is started to expire at this instant and the connection enters hold mode when it expires unless an LMP_not_accepted or LMP_hold_req is received by its LM before that point. If the LM receiving LMP_hold_req agrees to enter hold mode it returns LMP_accepted and starts a timer to expire at the hold instant. When this timer expires it enters hold mode.

When each LM exits from Hold mode it re-enables L2CAP transmission.



Sequence 29: Negotiation for hold mode.

9.3.16 Sniff mode

To enter sniff mode, master and slave negotiate a sniff interval T_{sniff} and a sniff offset, D_{sniff}, which specifies the timing of the sniff slots. The offset determines the time of the first sniff slot; after that the sniff slots follows periodically with the sniff interval T_{sniff}. To avoid problems with a clock wrap-around during the

Copyright © 2002 IEEE. All rights reserved.