

Wolfgang Rankl • Wolfgang Effing

Smart Card Handbook

Third Edition

 WILEY



Smart Card Handbook

Third Edition

Wolfgang Rankl and Wolfgang Effing
Giesecke & Devrient GmbH, Munich, Germany

Translated by
Kenneth Cox
Kenneth Cox Technical Translations, Wassenaar, The Netherlands



John Wiley & Sons, Ltd

Smart Card Handbook

Third Edition

Smart Card Handbook

Third Edition

Wolfgang Rankl and Wolfgang Effing
Giesecke & Devrient GmbH, Munich, Germany

Translated by
Kenneth Cox
Kenneth Cox Technical Translations, Wassenaar, The Netherlands



John Wiley & Sons, Ltd

Petitioner Micron Ex-1032, 0006

First published under the title *Handbuch der Chipkarten* by Carl Hanser Verlag
© Carl Hanser Verlag, Munich/FRG, 2002

All rights reserved.

Authorized translation from the 4th edition in the original German language
published by Carl Hanser Verlag, Munich/FRG.

Copyright © 2003 John Wiley & Sons Ltd, Baffins Lane, Chichester
West Sussex, PO19 1UD, England

National 01243 779777

International (+44) 1243 779777

Email (for orders and customer service enquiries): cs-books@wiley.co.uk

Visit our Home Page on www.wileyurope.com or www.wiley.com

All Rights Reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except under the terms of the Copyright, Designs and Patents Act 1988 or under the terms of a licence issued by the Copyright Licensing Agency Ltd, 90 Tottenham Court Road, London W1T 4LP, UK, without the permission in writing of the Publisher. Requests to the Publisher should be addressed to the Permissions Department,

John Wiley & Sons Ltd, The Atrium, Southern Gate, Chichester, West Sussex PO19 8SQ, England, or emailed to permreq@wiley.co.uk, or faxed to (+44) 1243 770571.

This publication is designed to provide accurate and authoritative information in regard to the subject matter covered. It is sold on the understanding that the Publisher is not engaged in rendering professional services. If professional advice or other expert assistance is required, the services of a competent professional should be sought.

Other Wiley Editorial Offices

John Wiley & Sons Inc., 111 River Street, Hoboken, NJ 07030, USA

Jossey-Bass, 989 Market Street, San Francisco, CA 94103-1741, USA

Wiley-VCH Verlag GmbH, Boschstr. 12, D-69469 Weinheim, Germany

John Wiley & Sons Australia Ltd, 33 Park Road, Milton, Queensland 4064, Australia

John Wiley & Sons (Asia) Pte Ltd, 2 Clementi Loop #02-01, Jin Xing Distripark, Singapore 129809

John Wiley & Sons Canada Ltd, 22 Worcester Road, Etobicoke, Ontario, Canada M9W 1L1

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic books.

Library of Congress Cataloging-in-Publication Data

Rankl, W. (Wolfgang)

[Handbuch der Chipkarten. English]

Smart card handbook / Wolfgang Rankl and Wolfgang Effing. – 3rd ed.

p. cm.

Includes bibliographical references and index.

ISBN 0-470-85668-8 (alk. paper)

1. Smart cards—Handbooks, manuals, etc. I. Effing, W. (Wolfgang) II. Title.

TK7895.S62R3613 2003

006 – dc22

2003062750

British Library Cataloguing in Publication Data

A catalogue record for this book is available from the British Library

ISBN 0-470-85668-8

Typeset in 10/12pt Times by TechBooks, New Delhi, India

Printed and bound in Great Britain by Antony Rowe Ltd, Chippenham Wiltshire

This book is printed on acid-free paper responsibly manufactured from sustainable forestry in which at least two trees are planted for each one used for paper production.

Contents

Preface to the Third Edition	xiii
Symbols and Notation	xv
Program Code Conventions	xvii
Abbreviations	xix
1 Introduction	1
1.1 The History of Smart Cards	2
1.2 Application Areas	5
1.2.1 Memory cards	6
1.2.2 Microprocessor cards	6
1.2.3 Contactless cards	8
1.3 Standardization	9
2 Types of Cards	15
2.1 Embossed Cards	15
2.2 Magnetic-stripe Cards	16
2.3 Smart Cards	18
2.3.1 Memory cards	19
2.3.2 Microprocessor cards	20
2.3.3 Contactless smart cards	21
2.4 Optical Memory Cards	23
3 Physical and Electrical Properties	27
3.1 Physical Properties	27
3.1.1 Card formats	28
3.1.2 Card components and security features	31
3.2 The Card Body	38
3.2.1 Card materials	40
3.2.2 Chip modules	42
3.3 Electrical Properties	52
3.3.1 Electrical connections	53
3.3.2 Supply voltage	55

3.3.3	Supply current	58
3.3.4	External clock	60
3.3.5	Data transmission	60
3.3.6	Activation and deactivation sequences	61
3.4	Smart Card Microcontrollers	62
3.4.1	Processor types	66
3.4.2	Memory types	70
3.4.3	Supplementary hardware	80
3.5	Contact-type Cards	91
3.6	Contactless Cards	93
3.6.1	Close-coupling cards: ISO/IEC 10536	101
3.6.2	Remote-coupling cards	107
3.6.3	Proximity integrated circuit(s) cards: ISO/IEC 14 443	108
3.6.4	Vicinity integrated circuits cards (ISO/IEC 15 693)	153
3.6.5	Test methods for contactless smart cards	153
4	Informatic Foundations	155
4.1	Structuring Data	156
4.2	Coding Alphanumeric Data	161
4.2.1	7-bit code	161
4.2.2	8-bit code	161
4.2.3	16-bit code (Unicode)	163
4.2.4	32-bit code (UCS)	163
4.3	SDL Notation	164
4.4	State Machines	165
4.4.1	Basic theory of state machines	166
4.4.2	Practical applications	166
4.5	Error Detection and Correction Codes	169
4.5.1	XOR checksums	171
4.5.2	CRC checksums	172
4.5.3	Reed–Solomon codes	174
4.5.4	Error correction	174
4.6	Data Compression	176
4.7	Cryptology	177
4.7.1	Symmetric cryptographic algorithms	182
4.7.2	Asymmetric cryptographic algorithms	189
4.7.3	Padding	199
4.7.4	Message authentication code and cryptographic checksum	201
4.8	Key Management	202
4.8.1	Derived keys	202
4.8.2	Key diversification	203
4.8.3	Key versions	203
4.8.4	Dynamic keys	203
4.8.5	Key parameters	204
4.8.6	Key management example	206
4.9	Hash Functions	208

	Contents	vii
4.10 Random Numbers		210
4.10.1 Generating random numbers		211
4.10.2 Testing random numbers		213
4.11 Authentication		216
4.11.1 Symmetric unilateral authentication		218
4.11.2 Symmetric mutual authentication		219
4.11.3 Static asymmetric authentication		222
4.11.4 Dynamic asymmetric authentication		223
4.12 Digital Signatures		225
4.13 Certificates		229
5 Smart Card Operating Systems		233
5.1 Historical Evolution of Smart Card Operating Systems		234
5.2 Fundamentals		237
5.3 Design and Implementation Principles		242
5.4 Completion		245
5.5 Memory Organization		249
5.6 Smart Card Files		252
5.6.1 File types		254
5.6.2 File names		257
5.6.3 File selection		261
5.6.4 EF file structures		263
5.6.5 File access conditions		267
5.6.6 File attributes		270
5.7 File Management		271
5.8 Sequential Control		279
5.9 Access to Resources in Accordance with ISO/IEC 7816-9		280
5.10 Atomic Operations		288
5.11 Open Platform		290
5.12 Downloadable Program Code		293
5.13 Executable Native Code		296
5.14 Open Platforms		302
5.14.1 Java Card		303
5.14.2 Multos		322
5.14.3 Basic Card		323
5.14.4 Windows for Smart Cards		323
5.14.5 Linux		324
5.15 The Small-OS Smart Card Operating System		326
6 Smart Card Data Transmission		371
6.1 The Physical Transmission Layer		373
6.2 Answer to Reset (ATR)		377
6.2.1 ATR characters		379
6.2.2 Practical examples of ATRs		389

6.3 Protocol Parameter Selection (PPS)	392
6.4 Data Transmission Protocols	396
6.4.1 Synchronous data transmission	397
6.4.2 The $T = 0$ transmission protocol	403
6.4.3 The $T = 1$ transmission protocol	409
6.4.4 The $T = 14$ transmission protocol (Germany)	419
6.4.5 The USB transmission protocol	420
6.4.6 Comparison of asynchronous transmission protocols	421
6.5 Message Structure: APDUs	421
6.5.1 Structure of the command APDU	422
6.5.2 Structure of the response APDU	424
6.6 Securing Data Transmissions	425
6.6.1 The authentic mode procedure	429
6.6.2 The combined mode procedure	430
6.6.3 Send sequence counter	432
6.7 Logical Channels	434
7 Smart Card Commands	435
7.1 File Selection Commands	439
7.2 Read and Write Commands	442
7.3 Search Commands	450
7.4 File Manipulation Commands	452
7.5 Identification Commands	453
7.6 Authentication Commands	457
7.7 Commands for Cryptographic Algorithms	462
7.8 File Management Commands	468
7.9 Commands for Managing Applets	474
7.10 Commands for Completing the Operating System	474
7.11 Commands for Hardware Testing	477
7.12 Commands for Data Transmission Protocols	481
7.13 Database Commands: SCQL	482
7.14 Commands for Electronic Purses	486
7.15 Commands for Credit and Debit Cards	489
7.16 Application-Specific Commands	490
8 Security Techniques	491
8.1 User Identification	491
8.1.1 Testing a secret number	493
8.1.2 Biometric methods	498
8.2 Smart Card Security	510
8.2.1 A classification of attacks and attackers	511
8.2.2 Attacks and defensive measures during development	517
8.2.3 Attacks and defensive measures during production	520
8.2.4 Attacks and defense measures while the card is in use	521
9 Quality Assurance and Testing	565
9.1 Card Body Tests	566

	Contents	ix
9.2 Microcontroller Hardware Tests		573
9.3 Evaluating and Testing Software		574
9.3.1 Evaluation		575
9.3.2 Test methods for software		581
9.3.3 Dynamic testing of operating systems and applications		589
10 The Smart Card Life Cycle		597
10.1 The Five Phases of the Smart Card Life Cycle		598
10.2 Phase 1 of the Life Cycle in Detail		600
10.2.1 Generating the operating system and producing the chip		600
10.2.2 Producing card bodies without integrated coils		612
10.2.3 Producing card bodies containing integrated coils		621
10.2.4 Combining the card body and the chip		628
10.3 Phase 2 of the Life Cycle in Detail		630
10.4 Phase 3 of the Life Cycle in Detail		638
10.5 Phase 4 of the Life Cycle in Detail		650
10.6 Phase 5 of the Life Cycle in Detail		652
11 Smart Card Terminals		655
11.1 Mechanical Properties		660
11.2 Electrical Properties		663
11.3 Security Technology		665
11.4 Connecting Terminals to Higher-Level Systems		667
11.4.1 PC/SC		667
11.4.2 OCF		671
11.4.3 MKT		672
11.4.4 MUSCLE		672
12 Smart Cards in Payment Systems		673
12.1 Payment Transactions using Cards		674
12.1.1 Electronic payments with smart cards		674
12.1.2 Electronic money		679
12.1.3 Basic system architecture options		681
12.2 Prepaid Memory Cards		684
12.3 Electronic Purses		685
12.3.1 The CEN EN 1546 standard		685
12.3.2 Common Electronic Purse Specifications (CEPS)		701
12.3.3 Proton		702
12.3.4 The Mondex system		703
12.4 The EMV Application		708
12.5 The Eurocheque System in Germany		714
13 Smart Cards in Telecommunications		723
13.1 Survey of Mobile Telecommunication Systems		727
13.1.1 Multiple-access methods		727
13.1.2 Cellular technology		730

13.1.3	Cell types	732
13.1.4	Bearer services	733
13.2	The GSM System	735
13.2.1	Specifications	737
13.2.2	System architecture and components	740
13.2.3	Important data elements	741
13.2.4	The subscriber identity module (SIM)	745
13.2.5	General Packet Radio System (GPRS)	786
13.2.6	Future developments	787
13.3	The UMTS System	789
13.4	Microbrowsers	794
13.5	The Wireless Identification Module (WIM)	802
13.6	Public Card Phones in Germany	804
14	Sample Applications	811
14.1	Contactless Memory Cards for Air Travel	811
14.2	Health Insurance Cards	814
14.3	Electronic Toll Systems	819
14.4	Digital Signatures	822
14.5	The PKCS #15 Signature Application	833
14.6	The FINEID Personal Identification Card	840
14.7	Tachosmart	840
15	Application Design	843
15.1	General Information and Characteristic Data	843
15.1.1	Microcontrollers	843
15.1.2	Applications	846
15.1.3	System considerations	848
15.1.4	Compliance with standards	850
15.2	Formulas for Estimating Processing Times	850
15.3	Timing Formulas for Typical Smart Card Commands	858
15.4	Typical Command Processing Times	860
15.5	Application Development Tools	864
15.6	Analyzing an Unknown Smart Card	868
15.7	Life-Cycle Models and Process Maturity	870
15.7.1	Life-cycle models	874
15.7.2	Process maturity	882
15.8	The Course of a Smart Card Project	885
15.9	Design Examples for Smart Card Applications	886
15.9.1	An electronic purse system for arcade games	888
15.9.2	Access control system	890
15.9.3	Testing the genuineness of a terminal	894
16	Appendix	897
16.1	Glossary	897
16.2	Related Reading	985

	Contents	xi
16.3 Literature		985
16.4 Annotated Directory of Standards and Specifications		994
16.5 Coding of Data Objects		1030
16.5.1 Data objects compliant with ISO/IEC 7816-4		1030
16.5.2 Data objects compliant with ISO/IEC 7816-6		1031
16.5.3 Data objects for chip manufacturers as specified by ISO/IEC 7816-6		1032
16.6 Registration Authorities for RIDs		1032
16.7 Selected RIDs		1032
16.8 Trade Fairs, Conferences and Conventions		1033
16.9 World Wide Web Addresses		1034
16.10 Characteristic Data and Tables		1044
16.10.1 ATR interval		1044
16.10.2 ATR parameter conversion tables		1044
16.10.3 Determining the data transmission rate		1046
16.10.4 Sampling times for serial data		1046
16.10.5 The most important smart card commands		1047
16.10.6 Summary of utilized instruction bytes		1051
16.10.7 Smart card command coding		1053
16.10.8 Smart card return codes		1056
16.10.9 Selected chips for memory cards		1058
16.10.10 Selected microcontrollers for smart cards		1060
Index		1067

Preface to the Third Edition

The English version of the *Smart Card Handbook* has now reached its third edition. In comparison with the previous edition, it has been considerably expanded and thoroughly updated to represent the current state of the technology. In this book, we attempt to cover all aspects of smart card technology, with the term 'technology' intentionally being understood in a very broad sense.

As in previous editions, we have remained true to our motto, 'better one sentence too many than one word too few'. We have described this ever-expanding subject in as much detail as possible. Even more examples, drawings and photographs have been added to make it easier to understand complicated relationships. The glossary has been enlarged to include many new terms covering all essential concepts related to smart cards, and it has been enhanced with cross-references. In many cases, it can provide a quick introduction to a particular subject. Altogether, these additions, extensions and improvements have resulted in a book that is more than three times as large as the first edition.

Here we can make a small comparison. Modern smart card operating systems currently comprise 120,000 lines of source code, which roughly corresponds to two books the size of the present edition. Even if you are not familiar with programming, you can readily appreciate how sophisticated these operating systems have become.

These small, colorful plastic cards with their semiconductor chips continue to spread from their original countries, Germany and France, throughout the world. In the coming years, this technology can be expected to outstrip all others, especially since it is still in its infancy and there is no end or consolidation in sight.

Smart card technology progresses in leaps and bounds, and we attempt to keep pace by publishing a new edition of the *Smart Card Handbook* every two to three years. The *Smart Card Handbook* represents the present state of technical knowledge, and in areas that are presently undergoing rapid change, we indicate possible paths of evolution. If certain things come to be seen differently at a later date, we can only remark that no one knows what the future will bring. Despite this, or perhaps just because of this, we welcome all comments, suggestions and proposed improvements, so that this book can continue to cover the subject of smart cards as completely as possible. Here we would like to explicitly thank the many attentive and interested readers who have pointed out unclear or ambiguous passages and errors. Once again, an errata list for this edition will be made available at www.wiley.co.uk/commstech/.

We would also like to thank our many friends and colleagues who have repeatedly offered valuable (and occasionally somewhat uncomfortable) suggestions for making this book better

and more complete. We would particularly like to thank Hermann Altschäfl, Peter van Elst, Klaus Finkenzeller, Thomas Graßl, Michael Schnellinger, Harald Vater and Dieter Weiß, as well as Kathryn Sharples at Wiley for her helpful support and Kenneth Cox for the translation.

Munich, June 2002

Wolfgang Rankl

[Rankl@gmx.net], [www.wiley.co.uk/commstech/]

Wolfgang Effing

[WEffing@gmx.net]

Symbols and Notation

General

- In accordance with ISO standards, the least-significant bit is always designated 1, rather than 0.
- In accordance with common usage, the term ‘byte’ refers to a sequence of eight bits and is equivalent to the term ‘octet’, which is often used in international standards.
- Length specifications for data, objects and all countable quantities are shown in decimal form, in agreement with the usual practice in smart card standards. All other values are usually shown as hexadecimal numbers and identified as such.
- The prefixes ‘kilo’ and ‘mega’ have the values of 1024 (2^{10}) and 1,048,576 (2^{20}), respectively, as is customary in the field of information technology.
- Depending on the context, binary values may not be explicitly identified as such.
- Commands used with smart cards are printed in upper-case characters (for example: SELECT FILE).

Representation of characters and numbers

42	decimal value
'00'	hexadecimal value
$^{\circ}0^{\circ}$, $^{\circ}1^{\circ}$	binary values
"ABC"	ASCII value
B_n	byte number n (for example: B1)
b_n	bit number n (for example: b2)
D_n	digit number n (for example: D3)

Logical functions

$\parallel$	concatenation (of data elements or objects)
$\oplus$	logical XOR operation

$\wedge$	logical AND operation
$\vee$	logical OR operation
$a \in M$	a is an element of the set M
$a \notin M$	a is not an element of the set M
$\{a, b, c\}$	the set of elements a, b, c

Cryptographic functions

enc $_{Xn}$ (K; D)	encryption using the algorithm X and an n -bit key, with the key K and the data D [for example: enc _{DES56} ('1...0'; 42)]
dec $_{Xn}$ (K; D)	decryption using the algorithm X and an n -bit key, with the key K and the data D [for example: dec _{IDEA128} ('1...0'; 42)]
$S :=$ sign $_{Xn}$ (K; D)	generating the signature S using the algorithm X and an n -bit key, with the key K and the data D [for example: sign _{RSA512} ('1...0'; "Wolf")]
$R :=$ verify $_{Xn}$ (K; S)	verifying the signature S using the algorithm X and an n -bit key, with the key K [for example: verify _{RSA512} ('1...9'; 42)]
Result = OK/NOK	

References

See: '...'	This is a cross-reference to another location in the book.
See also: '...'	This is a cross-reference to another location in the book where more information on the subject can be found.
[...]	This is a reference to a World Wide Web site listed in the Appendix.
[X Y]	This is a cross-reference to additional literature or standards listed in the Appendix. The format is: $X \in \{\text{surname of the first-named author}\}$ $Y \in \{\text{last two digits of the year of publication}\}$

Program Code Conventions

The syntax and semantics of the program code used in this book are based on the standard dialects of Basic. However, the use of explanations in natural language within a program listing is allowed, in order to promote the understandability of the code. Naturally, although this makes it easier for the reader to understand the code, it means that it is not possible to automatically convert the code into machine code. This compromise is justified by the significant improvement in readability that it provides.

<code>:=</code>	assignment operator
<code>::=</code>	definition operator
<code>=, !=, <, <=, >, =></code>	comparison operators
<code>+, -, ×, /</code>	arithmetic operators
<code>NOT</code>	logical not
<code>AND</code>	logical and
<code>OR</code>	logical or
<code> </code>	concatenation operator (e.g., coupling two byte strings)
<code>-</code>	end-of-line marker for multiline instructions
<code>// ...</code>	comment
<i>IO_Buffer</i>	variable (printed in italics)
Label:	jump or call location (printed in bold)
<code>GOTO ...</code>	jump
<code>CALL ...</code>	function call (subroutine call)
<code>RETURN</code>	return from a function (subroutine)
<code>IF ... THEN ...</code>	decision, type 1
<code>IF ... THEN ... ELSE ...</code>	decision, type 2
<code>SEARCH (...)</code>	search in a list; search string in parentheses
<code>STATUS</code>	query the result of a previously executed function call
<code>STOP</code>	terminate a process
<code>LENGTH (...)</code>	calculate the length
<code>EXIST</code>	test for presence (for example: an object or data element)
<code>WITH ...</code>	starts the definition of a variable or object as a reference
<code>END WITH</code>	ends the definition of a variable or object as a reference

Abbreviations

3DES	triple DES (<i>see glossary</i>)
3GPP	Third Generation Partnership Project (<i>see glossary</i>)
3GPP2	Third Generation Partnership Project 2 (<i>see glossary</i>)
A3, A5, A8	GSM algorithm 3, 5, 8 (<i>see glossary</i>)
AAM	application abstract machine
ABA	American Bankers' Association
ABS	acrylonitrile butadiene styrene
AC	access conditions (<i>see glossary</i>)
ACD	access control descriptor
ACK	acknowledge
ACM	accumulated call meter
ADF	application dedicated file
ADN	abbreviated dialing number
AES	Advanced Encryption Standard (<i>see glossary</i>)
AFI	application family identifier
AFNOR	<i>Association Française de Normalisation</i> (<i>see glossary</i>)
AGE	<i>Autobahngebührenerfassung</i> [motorway toll collection]
AGE	<i>automatische Gebührenerfassung</i> [automatic toll collection]
AID	application identifier (<i>see glossary</i>)
AM	access mode
Amd.	Amendment
AMPS	Advanced Mobile Phone Service (<i>see glossary</i>)
AND	logical AND operation
ANSI	American National Standards Institute (<i>see glossary</i>)
AoC	Advice of Charge
AODF	authentication object directory file
APACS	Association for Payment Clearing Services
APDU	application protocol data unit (<i>see glossary</i>)
A-PET	amorphous polyethylene terephthalate
API	application programming interface (<i>see glossary</i>)
AR	access rules
ARM	advanced RISC machine

ARR	access rule reference
ASC	application-specific command
ASCII	American Standard Code for Information Interchange
ASIC	application-specific integrated circuit
ASK	amplitude shift keying (<i>see glossary</i>)
ASN.1	Abstract Syntax Notation 1 (<i>see glossary</i>)
AT	attention
ATM	automated teller machine
ATQA	answer to request, type A
ATQB	answer to request, type B
ATR	answer to reset (<i>see glossary</i>)
ATS	answer to select
ATTRIB	PICC selection command, type B
AUX	auxiliary
B2A	business-to-administration (<i>see glossary</i>)
B2B	business-to-business (<i>see glossary</i>)
B2C	business-to-consumer (<i>see glossary</i>)
Basic	Beginners All Purpose Symbolic Instruction Code
BCD	binary-coded digit
Bellcore	Bell Communications Research Laboratories
BER	Basic Encoding Rules (<i>see glossary</i>)
BER-TLV	Basic Encoding Rules – tag, length, value
BEZ	Börsenevidenzzentrale [electronic purse clearing center for Geldkarte]
BGT	block guard time
BIN	bank identification number
bit	binary digit
BPF	basic processor functions
BPSK	binary phase-shift keying (<i>see glossary</i>)
BS	base station
BWT	block waiting time
CA	certification authority (<i>see glossary</i>)
CAD	chip accepting device (<i>see glossary</i>)
CAFE	Conditional Access for Europe (EU project)
CAMEL	Customized Applications for Mobile Enhanced Logic
CAP	card application (<i>see glossary</i>)
C-APDU	command APDU (<i>see glossary</i>)
CAPI	crypto API (application programming interface)
CASCADE	Chip Architecture for Smart Card and Portable Intelligent Devices
CASE	computer-aided software engineering
CAT	card application toolkit
CAVE	Cellular Authentication, Voice Privacy and Encryption
CBC	cipher block chaining

CC	Common Criteria (<i>see glossary</i>)
CCD	card-coupling device
CCD	charge-coupled device
CCITT	<i>Comité Consultatif International Télégraphique et Téléphonique</i> (now ITU) (<i>see glossary</i>)
CCR	chip-card reader
CCS	cryptographic checksum (<i>see glossary</i>)
CD	committee draft
CDF	certificate directory file
CDM	card-dispensing machine
CDMA	code division multiple access (<i>see glossary</i>)
CEN	<i>Comité Européen de Normalisation</i> (<i>see glossary</i>)
CENELEC	<i>Comité Européen de Normalisation Électrotechnique</i> [European Committee for Electronics Standardization]
CEPS	Common Electronic Purse Specifications, (previously: Common European Purse System) (<i>see glossary</i>)
CEPT	<i>Conférence Européenne des Postes et Télécommunications</i> (<i>see glossary</i>)
CFB	cipher feedback
CGI	common gateway interface
CHV	cardholder verification
CICC	contactless integrated circuit card
CID	card identifier
CISC	complex instruction set computer
CLA	class
CLK	clock
CLn	cascade level n, type A
CMM	capability maturity model (<i>see glossary</i>)
CMOS	complementary metal-oxide semiconductor
CMS	card management system
COS	chip operating system (<i>see glossary</i>)
COT	chip-on-tape (<i>see glossary</i>)
CRC	cyclic redundancy check (<i>see glossary</i>)
CRCF	clock rate conversion factor
CRT	Chinese remainder theorem
CRT	control reference template
Cryptoki	cryptographic token interface
CSD	circuit-switched data
C-SET	Chip-SET (secure electronic transaction)
CT	chipcard terminal
CT	card terminal
CT	cascade tag, type A
CT	cordless telephone
CT-API	chipcard terminal (CT) API (<i>see glossary</i>)
CTDE	cryptographic token data element
CTI	cryptographic token information

CTIO	cryptographic token information object
CVM	cardholder verification method
CWT	character waiting time
D	divisor
DAD	destination address
DAM	DECT authentication module (<i>see glossary</i>)
DAM	draft amendment
D-AMPS	Digital Advanced Mobile Phone Service (<i>see glossary</i>)
DAP	data authentication pattern
DB	database
DBF	database file
DBMS	database management system
DC/SC	Digital Certificates on Smart Cards
DCODF	data container object directory file
DCS	digital cellular system
DEA	data encryption algorithm (<i>see glossary</i>)
DECT	Digital Enhanced Cordless Telecommunications (previously: Digital European Cordless Telecommunications) (<i>see glossary</i>)
DER	Distinguished Encoding rules (<i>see glossary</i>)
DES	Data Encryption Standard (<i>see glossary</i>)
DF	dedicated file (also often: directory file) (<i>see glossary</i>)
DFA	differential fault analysis (<i>see glossary</i>)
DFÜ	<i>Datenfernübertragung</i> [data telecommunications]
DIL	dual in-line
DIN	<i>Deutsche Industrienorm</i> [German industrial standard]
DIS	draft international standard
DLL	dynamic link library
DMA	direct memory access
DO	data object
DoD	US Department of Defense
DOM	document object model
DOV	data over voice
DPA	differential power analysis (<i>see glossary</i>)
dpi	dots per inch
DR	divisor receive (PCD to PICC)
DRAM	dynamic random-access memory (<i>see glossary</i>)
DRI	divisor receive integer (PCD to PICC)
DS	divisor send (PICC to PCD)
DSA	digital signature algorithm
DSI	divisor send integer (PICC to PCD)
DTAUS	<i>Datenträgeraustausch</i> [data storage medium exchange]
DTD	document type definition
DTMF	dual-tone multiple-frequency
DVD	digital versatile disc
DVS	<i>Dateiverwaltungssystem</i> [file management system]

E	end of communication, type A
EBCDIC	extended binary-coded decimal interchange code
EC	elliptic curve
ec	Eurocheque
ECB	electronic codebook
ECBS	European Committee for Banking Standards (<i>see glossary</i>)
ECC	elliptic curve cryptosystems (<i>see glossary</i>)
ECC	error correction code (<i>see glossary</i>)
ECDSA	elliptic curve DSA
ECML	Electronic Commerce Modeling Language
ECTEL	European Telecom Equipment and Systems Industry
EDC	error detection code (<i>see glossary</i>)
EDGE	Enhanced Data Rates for GSM and TDMA Evolution (<i>see glossary</i>)
EDI	electronic data interchange
EDIFACT	electronic data interchange for administration, commerce and transport
EEPROM, E ² PROM	electrically erasable programmable read-only memory (<i>see glossary</i>)
EF	elementary file (<i>see glossary</i>)
EFF	Electronic Frontier Foundation
EFI	EF internal
EFTPOS	electronic fund transfer at point of sale
EFW	EF working
EGT	extra guard time, type B
EMV	Europay, MasterCard, Visa (<i>see glossary</i>)
EOF	end of frame, type B
EPROM	erasable programmable read-only memory (<i>see glossary</i>)
ESD	electrostatic discharge
ESPRIT	European Strategic Programme of Research and Development in Information Technology (EU project)
ETS	European Telecommunication Standard (<i>see glossary</i>)
ETSI	European Telecommunications Standards Institute (<i>see glossary</i>)
etu	elementary time unit (<i>see glossary</i>)
f	following page
FAR	false acceptance rate
FAT	file allocation table (<i>see glossary</i>)
FBZ	<i>Fehlbedienungszähler</i> [error counter, key fault presentation counter, retry counter] (<i>see glossary</i>)
fC	frequency of operating field (carrier frequency)
FCB	file control block
FCC	Federal Communications Commission
FCFS	first-come, first-serve
FCI	file control information
FCOS	flip chip on substrate

FCP	file control parameters
FD/CDMA	frequency division / code division multiple access (<i>see glossary</i>)
FDMA	frequency division multiple access (<i>see glossary</i>)
FDN	fixed dialing number
FDT	frame delay time, type A
FEAL	fast data encipherment algorithm
FET	field-effect transistor
ff	following pages
FID	file identifier (<i>see glossary</i>)
FIFO	first in, first out
FINEID	Finnish Electronic Identification Card
FIPS	Federal Information Processing Standard (<i>see glossary</i>)
FMD	file management data
FO	frame option
FPGA	field-programmable gate array (<i>see glossary</i>)
FPLMTS	Future Public Land Mobile Telecommunication Service (<i>see glossary</i>)
FRAM	ferroelectric random-access memory (<i>see glossary</i>)
FRR	false rejection rate
FS	file system
fS	frequency of subcarrier modulation
FSC	frame size for proximity card
FSCI	frame size for proximity card integer
FSD	frame size for coupling device
FSDI	frame size for coupling device integer
FSK	frequency-shift keying
FTAM	file transfer, access and management
FWI	frame waiting time integer
FWT	frame waiting time
FWTTEMP	temporary frame waiting time
gcd	greatest common denominator
GF	Galois fields
GGSN	gateway GPRS support node
GND	ground
GP	Global Platform (<i>see glossary</i>)
GPL	GNU public license
GPRS	General Packet Radio System (<i>see glossary</i>)
GPS	Global Positioning System
GSM	Global System for Mobile Communications (previously: Groupe Spécial Mobile) (<i>see glossary</i>)
GTS	GSM Technical Specification
GUI	graphical user interface
HAL	hardware abstraction layer (<i>see glossary</i>)
HBCI	Home Banking Computer Interface (<i>see glossary</i>)
HiCo	high coercivity

HLTA	Halt command, type A
HLTB	Halt command, type B
HSCSD	high-speed circuit switched data
HSM	hardware security module
HSM	high-security module
HSM	host security module
HTML	hypertext markup language
HTTP	hypertext transfer protocol
HV	Vickers hardness
HW	hardware
I/O	input/output
I ² C	inter-integrated circuit
IATA	International Air Transport Association
IBAN	international bank account number
I-block	information block
ICC	integrated-circuit card (<i>see glossary</i>)
ID	identifier
IDEA	international data encryption algorithm
IEC	International Electrotechnical Commission (<i>see glossary</i>)
IEEE	Institute of Electrical and Electronics Engineers
IEP	intersector electronic purse
IFD	interface device (<i>see glossary</i>)
IFS	information field size
IFSC	information field size for the card
IFSD	information field size for the interface device
IIC	institution identification codes
IMEI	international mobile equipment identity
IMSI	international mobile subscriber identity
IMT-2000	International Mobile Telecommunication 2000 (<i>see glossary</i>)
IN	intelligent network
INF	information field
INS	instruction
INTAMIC	International Association of Microcircuit Cards
IP	Internet protocol
IPES	Improved Proposed Encryption Standard
IrDA	Infrared Data Association
ISDN	Integrated Services Digital Network (<i>see glossary</i>)
ISF	internal secret file
ISIM	IP security identity module
ISO	International Organization for Standardization (<i>see glossary</i>)
IT	information technology
ITSEC	Information Technology Security Evaluation Criteria (<i>see glossary</i>)
ITU	International Telecommunications Union (<i>see glossary</i>)
IuKDG	<i>Informations- und Kommunikations-Gesetz</i> [German Information and Communications Act]

IV	initialization vector
IVU	in-vehicle unit
J2ME	Java 2 Micro Edition
JCF	Java Card Forum (<i>see glossary</i>)
JCRE	Java Card runtime environment (<i>see glossary</i>)
JCVM	Java Card virtual machine (<i>see glossary</i>)
JDK	Java development kit (<i>see glossary</i>)
JECF	Java Electronic Commerce Framework
JIT	just in time
JTC1	Joint Technical Committee One
JVM	Java virtual machine
K	key
Kc	ciphering key
KD	derived key
KFPC	key fault presentation counter
Ki	individual key
KID	identifier key
KM	master key
KS	session key
KVK	<i>Krankenversichertenkarte</i> [German medical insurance card]
LA	location area
LAN	local-area network
Lc	command length
LCSI	life cycle status indicator
Le	expected length
LEN	length
LFSR	linear-feedback shift register
LIFO	last in, first out
LND	last number dialed
LOC	lines of code
LoCo	low coercivity
LRC	longitudinal redundancy check
LSAM	load secure application module
lsb	least significant bit
LSB	least significant byte
M	month
MAC	message authentication code / data security code (<i>see glossary</i>)
MAOS	multi-application operating system
MBL	maximum buffer length
MBLI	maximum buffer length index
MCT	multifunctional card terminal (<i>see glossary</i>)
ME	mobile equipment
MEL	Multos Executable Language
MExE	mobile station execution environment (<i>see glossary</i>)

MF	master file (<i>see glossary</i>)
MFC	multi-function card, multifunctional smart card
MIME	Multipurpose Internet Mail Extensions
MIPS	million instructions per second
MLI	multiple laser image
MM	<i>moduliertes Merkmal</i> [modulated feature]
MMI	man-machine interface
MMS	multimedia messaging service
MMU	memory-management unit
MOC	matching-on-chip
MOO	mode of operation
MOSAIC	Microchip On-Surface and In-Card
MOSFET	metal-oxide semiconductor field-effect transistor
MoU	Memorandum of Understanding (<i>see glossary</i>)
MS	mobile station
msb	most significant bit
MSB	most significant byte
MSE	MANAGE SECURITY ENVIRONMENT
MTBF	mean time between failures
MUSCLE	Movement for the Use of Smart Cards in a Linux Environment
NAD	node address
NAK	negative acknowledgement
NBS	US National Bureau of Standards (<i>see glossary</i>)
NCSC	National Computer Security Center (<i>see glossary</i>)
NDA	nondisclosure agreement
NIST	US National Institute of Standards and Technology (<i>see glossary</i>)
nok	not OK
NPU	numeric processing unit (<i>see glossary</i>)
NRZ	non-return to zero
NSA	US National Security Agency (<i>see glossary</i>)
NU	not used
NVB	number of valid bits
OBU	onboard unit
ODF	object directory file
OFB	output feedback
OID	object identifier
OOK	on/off keying
OP	Open Platform (<i>see glossary</i>)
OR	logical OR operation
OS	operating system
OSI	Open Systems Interconnections
OTA	Open Terminal Architecture
OTA	over-the-air (<i>see glossary</i>)
OTASS	over-the-air SIM services
OTP	one-time password

OTP	one-time programmable
OTP	Open Trading Protocol
OVI	optically variable ink
P1, P2, P3	parameter 1, 2, 3
PA	power analysis
PB	procedure byte
PC	personal computer
PC	polycarbonate
PC/SC	personal computer / smart card (<i>see glossary</i>)
PCB	protocol control byte
PCD	proximity coupling device (<i>see glossary</i>)
PCMCIA	Personal Computer Memory Card International Association
PCN	personal communication networks
PCS	personal communication system
PDA	personal digital assistant
PES	proposed encryption standard
PET	polyethylene terephthalate
PETP	partially crystalline polyethylene terephthalate
PGP	Pretty Good Privacy
PICC	proximity ICC (<i>see glossary</i>)
PIN	personal identification number
PIX	proprietary application identifier extension
PKCS	public-key cryptography standards (<i>see glossary</i>)
PKI	public-key infrastructure (<i>see glossary</i>)
PLL	phase-locked loop
PLMN	public land mobile network (<i>see glossary</i>)
PM	person-month
POS	point of sale (<i>see glossary</i>)
POZ	<i>POS ohne Zahlungsgarantie</i> [POS without payment guarantee]
PP	protection profile (<i>see glossary</i>)
PPM	pulse position modulation
PPC	production planning and control
PPS	protocol parameter selection
prEN	<i>pre Norme Européenne</i> [preliminary European standard]
prETS	pre European Telecommunication Standard
PrKDF	private key directory file
PRNG	pseudorandom number generator (<i>see glossary</i>)
PROM	programmable read-only memory
PSAM	purchase secure application module
PSK	phase shift keying
PSO	PERFORM SECURITY OPERATION
PSTN	public switched telephone network (<i>see glossary</i>)
PTS	protocol type selection
PTT	<i>Postes Télégraphes et Téléphones</i> [post, telegraph and telephone]
Pub	publication

PUK	personal unblocking key (<i>see glossary</i>)
PuKDF	public key directory file
PUPI	pseudo-unique PICC identifier
PVC	polyvinyl chloride
PWM	pulse width modulation
RAM	random-access memory (<i>see glossary</i>)
R-APDU	response APDU (<i>see glossary</i>)
RATS	request to answer to select
REJ	reject
REQA	request command, type A
REQB	request command, type B
RES	resynchronization
RF	radio frequency
RFC	request for comment
RFID	radio frequency identification
RFU	reserved for future use
RID	record identifier
RID	registered application provider identifier
RIPE	RACE (EU project) integrity primitives evaluation
RIPE-MD	RACE integrity primitives evaluation message digest
RISC	reduced instruction set computer
RND	random number
RNG	random number generator
ROM	read-only memory (<i>see glossary</i>)
RS	Reed–Solomon
RSA	Rivest, Shamir and Adleman cryptographic algorithm
RTE	runtime environment
R-UIM	removable user identity module (<i>see glossary</i>)
S	start of communication
S@T	SIM Alliance Toolbox
S@T	SIM Alliance Toolkit
S@TML	SIM Alliance Toolbox Markup Language
SA	security attributes
SA	service area
SAD	source address
SAGE	Security Algorithm Group of Experts
SAK	select acknowledge
SAM	secure application module (<i>see glossary</i>)
SAT	SIM Application Toolkit (<i>see glossary</i>)
SC	security conditions
SC	smart card
SCC	smart card controller
SCMS	smart card management system
SCOPE	Smart Card Open Platform Environment (<i>see glossary</i>)
SCP	Smart Card Platform

SCQL	structured card query language
SCSUG	Smart Card Security Users Group
SDL	specification and description language
SDMA	space division multiple access (<i>see glossary</i>)
SE	security environment (<i>see glossary</i>)
SECCOS	Secure Chip Card Operating System (<i>see glossary</i>)
SEIS	Secured Electronic Information in Society
SEL	select code
SELECT	select command
SEMPER	Secure Electronic Marketplace for Europe (EU project)
SEPP	secure electronic payment protocol
SET	secure electronic transaction (<i>see glossary</i>)
SFGI	start-up frame guard time integer
SFGT	start-up frame guard time
SFI	short file identifier (<i>see glossary</i>)
SGSN	serving GPRS support node
S-HTTP	secure hypertext transfer protocol
SigG	<i>Signaturgesetz</i> [German electronic signature act] (<i>see glossary</i>)
SigV	<i>Signaturverordnung</i> [German electronic signature ordinance] (<i>see glossary</i>)
SIM	subscriber identity module (<i>see glossary</i>)
SIMEG	Subscriber Identity Module Expert Group (<i>see glossary</i>)
SKDF	secret key directory file
SM	secure messaging
SM	security mechanism
SMD	surface mounted device (<i>see glossary</i>)
SMG9	Special Mobile Group 9 (<i>see glossary</i>)
SMIME	Secure Multipurpose Internet Mail Extensions
SMS	short message service (<i>see glossary</i>)
SMSC	short message service center
SMS-PP	short message service point to point
SOF	start of frame
SPA	simple power analysis (<i>see glossary</i>)
SQL	structured query language
SQUID	superconducting quantum interference device
SRAM	static random-access memory (<i>see glossary</i>)
SRES	signed response
SS	supplementary service
SSC	send sequence counter
SSL	secure socket layer
SSO	single sign-on (<i>see glossary</i>)
STARCOS	Smart Card Chip Operating System (product of G+D)
STC	sub technical committee
STK	SIM Application Toolkit (<i>see glossary</i>)
STT	secure transaction technology
SVC	stored value card (product of Visa International)

SW	software
SW1, SW2	status word 1, 2
SWIFT	Society for Worldwide Interbank Financial Telecommunications
T	tag
TAB	tape-automated bonding
TACS	Total Access Communication System
TAL	terminal application layer
TAN	transaction number (<i>see glossary</i>)
TAR	toolkit application reference
tbd	to be defined
TC	trust center (<i>see glossary</i>)
TC	technical committee
TC	thermochrome
TCOS	Telesec Card Operating System
TCP	transport control protocol
TCP/IP	Transmission Control Protocol / Internet Protocol
TCSEC	Trusted Computer System Evaluation Criteria (<i>see glossary</i>)
TD/CDMA	time division / code division multiple access (<i>see glossary</i>)
TDES	triple DES (<i>see glossary</i>)
TDMA	time division multiple access (<i>see glossary</i>)
TETRA	Trans-European Trunked Radio (<i>see glossary</i>)
TLS	transport layer security
TLV	tag, length & value (<i>see glossary</i>)
TMSI	temporary mobile subscriber identity
TOE	target of evaluation (<i>see glossary</i>)
TPDU	transmission protocol data unit (<i>see glossary</i>)
TRNG	true random number generator (<i>see glossary</i>)
TS	technical specification
TTCN	tree-and-tabular combined notation
TTL	terminal transport layer
TTL	transistor-transistor logic
TTP	trusted third party (<i>see glossary</i>)
UART	universal asynchronous receiver/transmitter (<i>see glossary</i>)
UATK	UIM Application Toolkit
UCS	Universal Character Set (<i>see glossary</i>)
UI	user interface
UICC	universal integrated circuit card (<i>see glossary</i>)
UID	unique identifier
UIM	user identity module (<i>see glossary</i>)
UML	unified modeling language (<i>see glossary</i>)
UMTS	Universal Mobile Telecommunication System (<i>see glossary</i>)
URL	uniform resource locator (<i>see glossary</i>)
USAT	USIM application toolkit (<i>see glossary</i>)
USB	universal serial bus
USIM	universal subscriber identity module (<i>see glossary</i>)

USSD	unstructured supplementary services data
UTF	UCS transformation format
UTRAN	UMTS radio access network
VAS	value-added services (<i>see glossary</i>)
V _{cc}	supply voltage
VCD	vicinity coupling device
VEE	Visa Easy Entry (<i>see glossary</i>)
VKNR	<i>Versichertenkartennummer</i> [subscriber card number for German medical insurance]
VLSI	very large scale integration
VM	virtual machine (<i>see glossary</i>)
VOP	Visa Open Platform (<i>see glossary</i>)
V _{pp}	programming voltage
VSI	vertical system integration
W3C	World Wide Web Consortium
WAE	wireless application environment
WAN	wide-area network
WAP	wireless application protocol (<i>see glossary</i>)
WCDMA	wideband code division multiple access (<i>see glossary</i>)
WDP	wireless datagram protocol
WfSC	Windows for Smart Cards
WG	working group
WIG	wireless Internet gateway
WIM	wireless identification module (<i>see glossary</i>)
WML	wireless markup language (<i>see glossary</i>)
WORM	write once, read multiple
WSC	Windows for Smart Cards
WSP	wafer-scale package
WSP	wireless session protocol
WTAI	wireless telephony application interface
WTLS	wireless transport layer security
WTP	wireless transport protocol
WTX	waiting time extension
WTXM	waiting time extension multiplier
WUPA	wakeup command, type A
WUPB	wakeup command, type B
WWW	World Wide Web (<i>see glossary</i>)
XML	extensible markup language (<i>see glossary</i>)
XOR	logical exclusive-OR operation
Y	year
ZKA	<i>Zentraler Kreditausschuss</i> [Central Loans Committee] (<i>see glossary</i>)

1

Introduction

This book has been written for students, engineers and technically minded persons who want to learn more about smart cards. It attempts to cover this broad topic as completely as possible, in order to provide the reader with a general understanding of the fundamentals and the current state of the technology.

We have put great emphasis on a practical approach. The wealth of pictures, tables and references to real applications is intended to help the reader become familiar with the subject rather more quickly than would be possible with a strictly technical presentation. This book is thus intended to be useful in practice, rather than technically complete. For this reason, descriptions have been kept as concrete as possible. In places where we were faced with a choice between technical accuracy and ease of understanding, we have tried to strike a happy medium. Whenever this proved to be impossible, we have always given preference to ease of understanding.

The book has been written so that it can be read in the usual way, from front to back. We have tried to avoid forward references as much as possible. The designs of the individual chapters, in terms of structure and content, allow them to be read individually without any loss of understanding. The comprehensive index and the glossary allow this book to be used as a reference work. If you want to know more about a specific topic, the references in the text and the annotated directory of standards will help you find the relevant documents.

Unfortunately, a large number of abbreviations have become established in smart card technology, as in so many other areas of technology and everyday life. This makes it particularly difficult for newcomers to become familiar with the subject. We have tried to minimize the use of these cryptic and frequently illogical abbreviations. Nevertheless, we have often had to choose a middle way between internationally accepted smart card terminology used by specialists and common terms more easily understood by laypersons. If we have not always succeeded, the extensive list of abbreviations at the front of the book should at least help overcome any barriers to understanding, which we hope will be short-lived. An extensive glossary in the final chapter of the book explains the most important technical concepts and supplements the list of abbreviations.

An important feature of smart cards is that their properties are strongly based on international standards. This is fundamentally important with regard to the usually compulsory need for interoperability. Unfortunately, these standards are often difficult to understand, and in

some critical places they require outright interpretation. Sometimes only the members of the associated standardization group can explain the intention of certain sections. In such cases, the *Smart Card Handbook* attempts to present the understanding that is generally accepted in the smart card industry. Nevertheless, the relevant standards are still the ultimate authority, and in such cases they should always be consulted.

1.1 THE HISTORY OF SMART CARDS

The proliferation of plastic cards started in the USA in the early 1950s. The low price of the synthetic material PVC made it possible to produce robust, durable plastic cards that were much more suitable for everyday use than the paper and cardboard cards previously used, which could not adequately withstand mechanical stresses and climatic effects.

The first all-plastic payment card for general use was issued by the Diners Club in 1950. It was intended for an exclusive class of individual, and thus also served as a status symbol, allowing the holder to pay with his or her 'good name' instead of cash. Initially, only the more select restaurants and hotels accepted these cards, so this type of card came to be known as a 'travel and entertainment' card.

The entry of Visa and MasterCard into the field led to a very rapid proliferation of 'plastic money' in the form of credit cards. This occurred first in the USA, with Europe and the rest of the world following a few years later. Today, credit cards allow travelers to shop without cash everywhere in the world. A cardholder is never at a loss for means of payment, yet he or she avoids exposure to the risk of loss due to theft or other unpredictable hazards, particularly while traveling. Using a credit card also eliminates the tedious task of exchanging currency when traveling abroad. These unique advantages helped credit cards become rapidly established throughout the world. Many hundreds of millions of cards are produced and issued annually.

At first, the functions of these cards were quite simple. They served as data storage media that were secure against forgery and tampering. General information, such as the card issuer's name, was printed on the surface, while personal data elements, such as the cardholder's name and the card number, were embossed. Many cards also had a signature panel where the cardholder could sign his or her name for reference. In these first-generation cards, protection against forgery was provided by visual features, such as security printing and the signature panel. Consequently, the system's security depended quite fundamentally on the quality and conscientiousness of the persons responsible for accepting the cards. However, this did not represent an overwhelming problem, due to the card's initial exclusivity. With the increasing proliferation of card use, these rather rudimentary features no longer proved sufficient, particularly since threats from organized criminals were growing apace.

Increasing handling costs for merchants and banks made a machine-readable card necessary, while at the same time, losses suffered by card issuers as the result of customer insolvency and fraud grew from year to year. It became apparent that the security features for protection against fraud and manipulation, as well as the basic functions of the card, had to be expanded and improved.

The first improvement consisted of a magnetic stripe on the back of the card, which allowed digital data to be stored on the card in machine-readable form as a supplement to the visual information. This made it possible to minimize the use of paper receipts, which were previously essential, although the customer's signature on a paper receipt was still required in traditional credit card applications as a form of personal identification. However, new approaches that did

not require paper receipts could also be devised. This made it possible to finally achieve the long-standing objective of replacing paper-based transactions by electronic data processing. This required a different method to be used for user identification, which previously employed the user's signature. The method that has come into widespread general use involves a secret personal identification number (PIN) that is compared with a reference number. The reader is surely familiar with this method from using bank machines (automated teller machines). Embossed cards with magnetic stripes are still the most commonly used types of cards for financial transactions.

However, magnetic-stripe technology has a crucial weakness, which is that the data stored on the stripe can be read, deleted and rewritten at will by anyone with access to the necessary equipment. It is thus unsuitable for storing confidential data. Additional techniques must be used to ensure confidentiality of the data and prevent manipulation of the data. For example, the reference value for the PIN could be stored in the terminal or host system in a secure environment, instead of on the magnetic stripe. Most systems that employ magnetic-stripe cards thus use online connections to the system's host computer for reasons of security, even though this generates significant costs for the necessary data transmissions. In order to reduce costs, it is necessary to find solutions that allow card transactions to be executed offline without endangering the security of the system.

The development of the smart card, combined with the expansion of electronic data-processing systems, has created completely new possibilities for devising such solutions. Enormous progress in microelectronics in the 1970s made it possible to integrate data storage and processing logic on a single silicon chip measuring a few square millimetres. The idea of incorporating such an integrated circuit into an identification card was contained in a patent application filed by the German inventors Jürgen Dethloff and Helmut Grötrup as early as 1968. This was followed in 1970 by a similar patent application by Kunitaka Arimura in Japan. However, the first real progress in the development of smart cards came when Roland Moreno registered his smart card patents in France in 1974. It was only then that the semiconductor industry was able to supply the necessary integrated circuits at acceptable prices. Nevertheless, many technical problems still had to be solved before the first prototypes, some of which contained several integrated circuit chips, could be transformed into reliable products that could be manufactured in large numbers with adequate quality at a reasonable cost. Since the basic inventions in smart card technology originated in Germany and France, it is not surprising that these countries played the leading roles in the development and marketing of smart cards.

The great breakthrough was achieved in 1984, when the French PTT (postal and telecommunications services agency) successfully carried out a field trial with telephone cards. In this field trial, smart cards immediately proved to meet all expectations with regard to high reliability and protection against manipulation. Significantly, this breakthrough for smart cards did not come in an area where traditional cards were already used, but in a new application. Introducing a new technology in a new application has the great advantage that compatibility with existing systems does not have to be taken into account, so the capabilities of the new technology can be fully exploited.

A pilot project was conducted in Germany in 1984–85, using telephone cards based on several technologies. Magnetic-stripe cards, optical-storage (holographic) cards and smart cards were used in comparative tests. Smart cards proved to be the winners in this pilot study. In addition to a high degree of reliability and security against manipulation, smart card technology promised the greatest degree of flexibility for future applications. Although the older but less expensive EPROM technology was used in the French telephone card chips,

more recent EEPROM chips were used from the start in the German telephone cards. The latter type of chip does not need an external programming voltage. An unfortunate consequence is that the French and German telephone cards are mutually incompatible. It appears that even after the introduction of the euro, French and German telephone cards will remain unusable in each other's country of origin for at least a while.

Further developments followed the successful trials of telephone cards, first in France and then in Germany, with breathtaking speed. By 1986, several million 'smart' telephone cards were in circulation in France alone. The total rose to nearly 60 million in 1990, and to several hundred million worldwide in 1997. Germany experienced similar progress, with a time lag of about three years. These systems were marketed throughout the world after the successful introduction of the smart card public telephone in France and Germany. Telephone cards incorporating chips are currently used in more than 50 countries.

The integrated circuits used in telephone cards are relatively small, simple and inexpensive memory chips with specific security logic that allows the card balance to be reduced while protecting it against manipulation. Microprocessor chips, which are significantly larger and more complex, were first used in large numbers in telecommunications applications, specifically for mobile telecommunications. In 1988, the German Post Office acted as a pioneer in this area by introducing a modern microprocessor card using EEPROM technology as an authorization card for the analog mobile telephone network (C-Netz). The reason for introducing such cards was an increasing incidence of fraud with the magnetic-stripe cards used up to that time. For technical reasons, the analog mobile telephone network was limited to a relatively small number of subscribers (around one million), so it was not a true mass market for microprocessor cards. However, the positive experience gained from using smart cards in the analog mobile telephone system was decisive for the introduction of smart cards into the digital GSM network. This network was put into service in 1991 in various European countries and has presently expanded over the entire world, with over 600 million subscribers in more than 170 countries.

Progress was significantly slower in the field of bank cards, in part due to their greater complexity compared with telephone cards. These differences are described in detail in the following chapters. Here we would just like to remark that the development of modern cryptography has been just as crucial for the proliferation of bank cards as developments in semiconductor technology.

With the general expansion of electronic data processing in the 1960s, the discipline of cryptography experienced a sort of quantum leap. Modern hardware and software made it possible to implement complex, sophisticated mathematical algorithms that allowed previously unparalleled levels of security to be achieved. Moreover, this new technology was available to everyone, in contrast to the previous situation in which cryptography was a covert science in the private reserve of the military and secret services. With these modern cryptographic procedures, the strength of the security mechanisms in electronic data-processing systems could be mathematically calculated. It was no longer necessary to rely on a highly subjective assessment of conventional techniques, whose security essentially rests on the secrecy of the procedures used.

The smart card proved to be an ideal medium. It made a high level of security (based on cryptography) available to everyone, since it could safely store secret keys and execute cryptographic algorithms. In addition, smart cards are so small and easy to handle that they can be carried and used everywhere by everybody in everyday life. It was a natural idea to attempt to use these new security features for bank cards, in order to come to grips with the security risks arising from the increasing use of magnetic-stripe cards.

The French banks were the first to introduce this fascinating technology in 1984, following a trial with 60,000 cards in 1982–83. It took another 10 years before all French bank cards incorporated chips. In Germany, the first field trials took place in 1984–85, using a multifunctional payment card incorporating a chip. However, the *Zentrale Kreditausschuss* (ZKA), which is the coordinating committee of the leading German banks, did not manage to issue a specification for multifunctional Eurocheque cards incorporating chips until 1996. In 1997, all German savings associations and many banks issued the new smart cards. In the previous year, multifunctional smart cards with POS functions, an electronic purse and optional value-added services were issued in all of Austria. This made Austria the first country in the world to have a nationwide electronic purse system.

An important milestone for the future worldwide use of smart cards for making payments was the completion of the EMV specification, which was a product of the joint efforts of Europay, MasterCard and Visa. The first version of this specification was published in 1994. It contained detailed descriptions of credit cards incorporating microprocessor chips, and it guaranteed the mutual compatibility of the future smart cards of the three largest credit card organizations.

Electronic purse systems have proven to be another major factor in promoting the international use of smart cards for financial transactions. The first such system, called Danmønt, was put into operation in Denmark in 1992. There are currently more than 20 national systems in use in Europe alone, many of which are based on the European EN 1546 standard. The use of such systems is also increasing outside of Europe. In the USA, where smart-card systems have had a hard time becoming established, Visa experimented with a smart-card purse during the 1996 Olympic Summer Games in Atlanta. Payments via the Internet offer a new and promising application area for electronic purses. However, the problems associated with making small payments securely but anonymously throughout the world via the public Internet have not yet been solved in a satisfactory manner. Smart cards could play a decisive role in providing an answer to these problems. Besides this, smart cards could play an important role in introducing electronic signatures. Several European countries have initiated the introduction of electronic signature systems after a legal basis for the use of electronic signatures was provided by approval of a European directive regarding electronic signatures in 1999.

As the result of another application, almost every German citizen now possesses a smart card. When health insurance cards incorporating chips were introduced, more than 70 million smart cards were issued to all persons enrolled in the national health insurance plan. Presently, smart cards are being used in the health-care sector in many countries.

The smart card's high degree of functional flexibility, which even allows programs for new applications to be added to a card already in use, has opened up completely new application areas extending beyond the boundaries of traditional card uses.

Smart cards are also being used as 'electronic tickets' for local public transport in many cities throughout the world. Contactless smart cards are usually used for such applications, since they are particularly convenient and user friendly.

1.2 APPLICATION AREAS

As can be seen from the historical summary, the potential applications for smart cards are extremely diverse. With the steadily increasing storage and processing capacities of available integrated circuits, the range of potential applications is constantly being expanded. Since it is

impossible to describe all of these applications in detail within the confines of this book, a few typical examples must serve to illustrate the basic properties of smart cards. This introductory chapter is only meant to provide an initial overview of the functional versatility of these cards. Several typical applications are described in detail in Chapters 12, 13 and 14.

To make this overview easier to follow, it is helpful to divide smart cards into two categories: memory cards and microprocessor cards.

1.2.1 Memory cards

The first smart cards used in large quantities were memory cards for telephone applications. These cards are prepaid, with the value stored electronically in the chip being decreased by the amount of the call charge each time the card is used. Naturally, it is necessary to prevent the user from subsequently increasing the stored value, which could easily be done with a magnetic-stripe card. With such a card, all the user would have to do is record the data stored at the time of purchase and rewrite them to the magnetic stripe after using the card. The card would then have its original value and could be reused. This type of manipulation, known as ‘buffering’, is prevented in smart phone cards by security logic in the chip that makes it impossible to erase a memory cell once it has been written. The reduction of the card balance by the number of charge units used is thus irreversible.

This type of smart card can naturally be used not only for telephone calls, but also whenever goods or services are to be sold against prior payment without the use of cash. Examples of possible uses include local public transport, vending machines of all types, cafeterias, swimming pools, car parks and so on. The advantage of this type of card lies in its simple technology (the surface area of the chip is typically only a few square millimeters), and hence its low cost. The disadvantage is that the card cannot be reused once it is empty, but must be discarded as waste – unless it ends up in a card collection.

Another typical application of memory cards is the German health insurance card, which has been issued since 1994 to all persons enrolled in the national health insurance plan. The information previously written on the patient’s card is now stored in the chip and printed or laser-engraved on the card. Using a chip for data storage makes the cards machine-readable using simple equipment.

In summary, memory-type smart cards have limited functionality. Their integrated security logic makes it possible to protect stored data against manipulation. They are suitable for use as prepaid cards or identification cards in systems where low cost is a primary consideration.

1.2.2 Microprocessor cards

As already noted, microprocessor cards were first used in the form of bank cards in France. Their ability to securely store private keys and execute modern cryptographic algorithms made it possible to implement highly secure offline payment systems.

Since the microprocessor built into the card is freely programmable, the functionality of microprocessor cards is restricted only by the available storage space and the capacity of the processor. The only limits to the designer’s imagination when implementing smart card systems are thus technological, and they are extended enormously with each new generation of integrated circuits.

Following a drastic reduction in the cost of smart cards in the early 1990s due to mass production, new applications have been introduced year after year. The use of smart cards with mobile telephones has been especially important for their international proliferation. After being successfully tested in the German national C-Netz (analog mobile telephone network) for use in mobile telephones, smart cards were prescribed as the access medium for the European digital mobile telephone system (GSM). In part, this was because smart cards allowed a high degree of security to be achieved for accessing the mobile telephone network. At the same time, they provided new possibilities and thus major advantages in marketing mobile telephones, since they made it possible for network operators and service providers to sell telephones and services separately. Without the smart card, mobile telephones would certainly not have spread so quickly across Europe or developed into a worldwide industry standard.

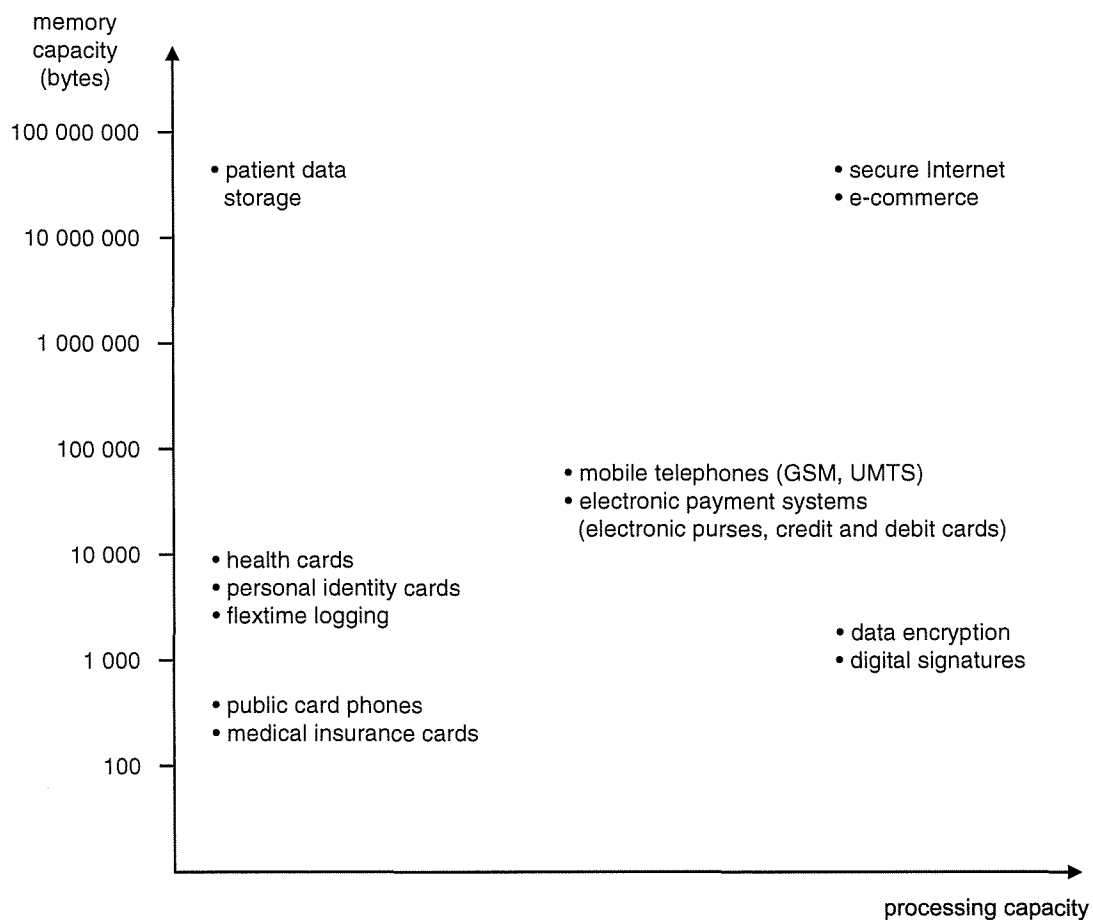


Figure 1.1 Typical smart card application areas, showing the required storage capacity and arithmetic processing capacity

Possible applications for microprocessor cards include identification, access control systems for restricted areas and computers, secure data storage, electronic signatures and electronic purses, as well as multifunctional cards incorporating several applications in a single

card. Modern smart-card operating systems also allow new applications to be loaded into a card after it has already been issued to the user, without compromising the security of the various applications. This new flexibility opens up completely new application areas. For example, personal security modules are indispensable if Internet commerce and payments are to be made trustworthy. Such security modules could securely store personal keys and execute high-performance cryptographic algorithms. These tasks can be performed in an elegant manner by a microprocessor with a cryptographic coprocessor. Specifications for secure Internet applications using smart cards are currently being developed throughout the world. Within a few years, we can expect to see every PC equipped with a smart-card interface.

In summary, the essential advantages of microprocessor cards are large storage capacity, the ability to securely store confidential data and the ability to execute cryptographic algorithms. These advantages make a wide range of new applications possible, in addition to the traditional bank card application. The potential of smart cards is by no means yet exhausted, and furthermore, it is constantly being expanded by progress in semiconductor technology.

1.2.3 Contactless cards

Contactless cards, in which energy and data are transferred without any electrical contact between the card and the terminal, have achieved the status of commercial products in the last few years. Presently, both memory cards and microprocessor cards are available as contactless cards. Although contactless microprocessor cards can usually work at a distance of only a few centimeters from the terminal, contactless memory cards can be used up to a meter away from the terminal. This means that such cards do not necessarily have to be held in the user's hand during use, but can remain in the user's purse or wallet.

Contactless cards are thus particularly suitable for applications in which persons or objects should be quickly identified. Sample applications are:

- access control,
- local public transportation,
- ski passes,
- airline tickets,
- baggage identification.

However, there are also applications where operation over a long distance could cause problems and should thus be prevented. A typical example is an electronic purse. A declaration of intent on the part of the cardholder is normally required to complete a financial transaction. This confirms the amount of the payment and the cardholder's agreement to pay. With a contactless card, this declaration takes the form of inserting the card in the terminal and confirming the indicated amount using the keypad. If contactless payments over relatively long distances were possible, a 'con artist' could remove money from the electronic purse without the knowledge of the cardholder. Dual-interface cards (sometimes called 'combicards') offer a possible solution to this problem. These cards combine contact and contactless interfaces in a single card. Such

a card can communicate with the terminal via either its contact interface or its contactless interface, according to what is desired.

There is great interest in contactless cards in the field of local public transportation. If the smart cards presently used in payment systems, which are generally contact-type cards, can have their functionality extended to include acting as electronic tickets with contactless interfaces, transportation operators could use the existing infrastructure and cards of the credit card industry.

1.3 STANDARDIZATION

The prerequisite for the worldwide penetration of smart cards into everyday life, such as their current use in Germany in the form of telephone cards, health insurance cards and bank cards, has been the creation of national and international standards. Due to the special significance of such standards, in this book we repeatedly refer to currently applicable standards and those that are in preparation. Why are standards so important for expanding the use of smart cards?

A smart card is normally one component of a complex system. This means that the interfaces between the card and the rest of the system must be precisely specified and matched to each other. Of course, this could be done for each system on a case-by-case basis, without regard to other systems. However, this would mean that a different type of smart card would be needed for each system. Users would thus have to carry a separate card for each application. In order to avoid this, an attempt has been made to generate application-independent standards that allow multifunctional cards to be developed.

Since the smart card is usually the only component of the system that the user holds in his or her hand, it is enormously important with regard to the recognition and acceptance of the entire system. Nonetheless, from a technical and organizational perspective, the smart card is usually only the tip of the iceberg, since complex systems (which are usually networked) are quite often hidden behind the card terminal, and it is these systems that make the services possible in the first place.

Let us take telephone cards as an example. In technical terms, they are fairly simple objects. By themselves, they are almost worthless, except perhaps as collector's items. Their true function, which is to allow public telephones to be used without coins, can be realized only after umpteen thousand card phones have been installed throughout a region and connected to a network. The large investments required for this can only be justified if the long-term viability of the system is ensured by appropriate standards and specifications. Standards are also an indispensable prerequisite for multifunctional smart cards used for several applications, such as telephony, electronic purses, electronic tickets and so on.

What are standards

This question is not as trivial as it may appear at first glance, since the terms 'standard' and 'specification' are often used fairly indiscriminately. To make things clear, let us consider the ISO/IEC definition:

Standard: a document that is produced by consensus and adopted by a recognized organization, and which, for general and recurring applications, defines rules, guidelines or features for

activities or their results, with the objective of achieving an optimum degree of regulation in a given context.

Note: standards should be based on the established results of science, technology and experience, and their objective should be the promotion of optimized benefits for society.

International standards should thus help make life easier and increase the reliability and usefulness of products and services. In order to avoid confusion, ISO/IEC have also defined the term 'consensus' as follows:

Consensus: general agreement, characterized by the absence of continuing objections to essential elements on the part of any significant portion of the interested parties, and achieved by a procedure that attempts to consider the views of all relevant parties and address all counter-arguments.

Note: consensus does not necessarily mean unanimity.

Although unanimity is not required for consensus, the democratic process naturally takes time. This is in particular due to the fact that it is necessary to consider not only the views of the technical specialists, but also the views of all relevant parties, since the objective of a standard is the promotion of optimum benefits for the whole of society. Hence, the preparation of an ISO or CEN standard usually takes several years. A frequent consequence of the slowness of this process is that a small group of interested parties, such as commercial firms, generates its own specification ('industry standard') in order to hasten the development of new systems. This is particularly true in the field of information technology, which is characterized by especially fast development and correspondingly short innovation cycles. Although industry standards and specifications have the advantage that they can be developed significantly faster than 'true' standards, they carry the risk of ignoring the interests of the parties that are not involved in their development. For this reason, ISO attempts to create possibilities for retroactively incorporating significant publicly accessible specifications into international standards.

What does ISO/IEC mean?

The ISO/IEC standards are especially significant for smart cards, since they define the basic properties of smart cards. What lies behind the abbreviations 'ISO' and 'IEC'? 'ISO' stands for the International Organization for Standardization, while 'IEC' stands for the International Electrotechnical Commission.

The International Organization for Standardization (ISO) is a worldwide association of around 100 national standards agencies, with one per country. ISO was founded in 1948 and is a non-national organization. Its task is to promote the development of standards throughout the world, with the objective of simplifying the international exchange of goods and services and developing cooperation in the fields of science, technology and economy. The results of the activities of ISO are agreements that are published as ISO standards.

Incidentally, 'ISO' is not an abbreviation (the abbreviation of the official name would of course be 'IOS'). The name 'ISO' is derived from the Greek word *isos*, which means 'equal' or 'the same'. The prefix *iso-*, derived from the Greek *isos*, is commonly used in

the three official languages of ISO (English, French and Russian), as well as in many other languages.

As already noted, the members of ISO are the national standards bodies of the individual countries, and only one such body per country is allowed to be a member. The member organizations have four basic tasks, as follows:

- to inform potentially interested parties in their own countries about relevant activities and possibilities of international standardization,
- to form national opinions on a democratic basis and represent these opinions in international negotiations,
- to set up a secretariat for ISO committees in which the country has a particular interest,
- to pay the country's financial contribution in support of the central ISO organization.

The IEC is a standardization organization whose scope of activity covers the areas of electrical technology and electronics. The first card standards were published by the IEC. After the introduction of smart cards, a difference of focus arose between the ISO and the IEC. In order to avoid duplication of effort, standards are developed in joint technical committees and published as ISO/IEC standards.

How is an ISO standard generated?

The need for a standard is usually reported to a national standards organization by an industrial sector. The national organization then proposes this to ISO as a new working theme. If the proposal is accepted by the responsible working group, which consists of technical experts from countries that are interested in the theme, the first thing that is done is to define the objective of the future standard.

After agreement has been reached with regard to the technical aspects to be considered in the standard, the detailed specifications of the standard are discussed and negotiated among the various countries. This is the second phase in the development of the standard. The objective of this phase is to arrive at a consensus of all participating countries, if possible. The outcome of this phase is a 'Draft International Standard' (DIS).

The final phase consists of a formal vote on the proposed standard. Acceptance of a standard requires the approval of two-thirds of the ISO members that actively participated in drafting the standard, as well as three-quarters of all members participating in the vote. Once the text has been accepted, it is published as an ISO standard.

To prevent standards from becoming outdated as the result of ongoing development, ISO rules state that standards should be reviewed, and if necessary revised, after an interval of at most five years.

Cooperation with the IEC and the CEN

ISO is not the only international standards organization. In order to avoid duplication of effort, ISO cooperates closely with the IEC (International Electrotechnical Commission). The areas

of responsibility are defined as follows: the IEC covers the fields of electrical technology and electronics, while ISO covers all other fields. Combined working groups are formed to deal with themes of common interest, and these groups produce combined ISO/IEC standards. Most standards for smart cards belong to this category.

ISO and the European standardization committee CEN (*Comité Européen de Normalisation*) also agree on rules for the development of standards that are recognized as both European and international standards. This leads to time and cost savings.

International standardization of smart cards

International standards for smart cards are developed under the auspices of ISO/IEC, and on the European level by the CEN. The major industrial countries are represented in all relevant committees, and they generally also maintain 'mirror' committees in the form of national working groups and voting committees. In Germany, this responsibility is borne by the DIN. Figure 1.2 shows an overview of the structure of the relevant ISO and IEC working groups and the standards for which they are responsible.

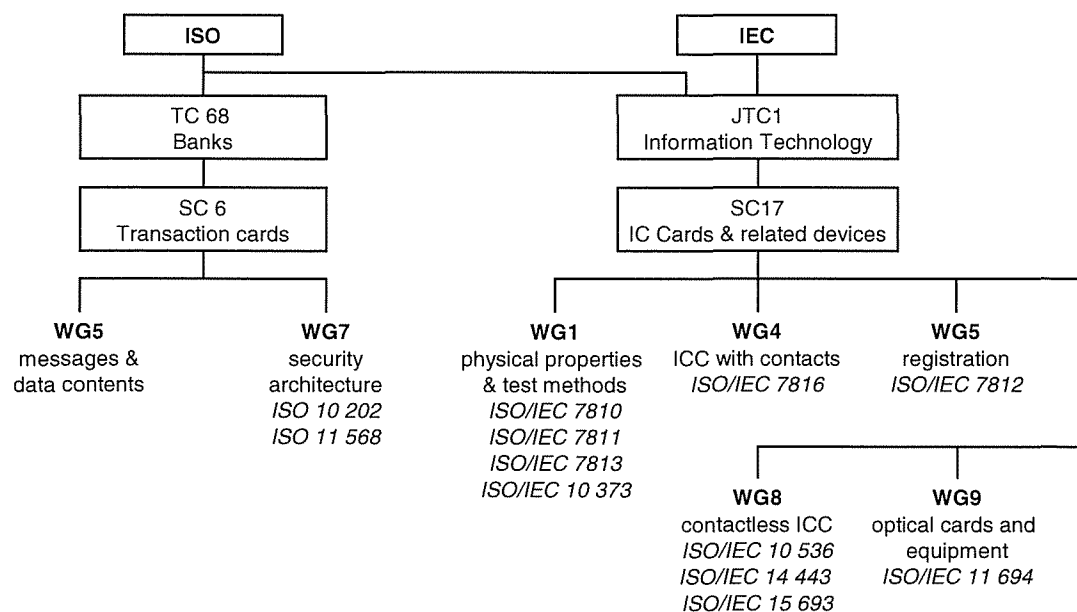


Figure 1.2 Overview and organization of the working groups for international smart card standards

As can be seen, there are two technical committees that are concerned with the standardization of smart cards. The first is ISO TC68/SC6, which is responsible for the standardization of cards used in the financial transaction area, while the second is ISO/IEC JTC1/SC17, which is responsible for general applications. This division has historical roots, since the first international applications were for identification cards used for financial transactions. The number of applications has naturally increased enormously since then, so the general standards, which are

looked after by the SC17 committee, have taken on greater significance. The standards specifically related to financial transactions can thus be regarded as a subset of the general standards. Brief descriptions of the standards listed in Figure 1.2, including their current status, can be found in Chapter 16, 'Appendix'.

Within CEN, the general subject of smart cards is dealt with by the TC224 committee ('Machine-readable Cards, Related Device Interfaces and Procedures'). The activities of CEN complement those of ISO. ISO standards are adopted as CEN standards where possible, which means they must be translated into the three official CEN languages (English, French and German). They may also be enlarged or reduced as necessary to comply with specific European conditions. The CEN working groups also produce application-specific standards, which would not be possible as such within ISO.

An additional European standardization body, the European Telecommunications Standards Institute (ETSI), has made a significant contribution to the widespread international use of smart cards. ETSI is the standardization body of the European telecommunications companies and telecommunication industry. The GSM 11.11 family of standards specifies the interface between the smart card (referred to as the 'subscriber identity module' (SIM) in the GSM system) and the mobile telephone. This family of standards is based on the ISO/IEC standards. With the international proliferation of GSM systems beyond the boundaries of Europe, the ETSI standards have become highly important for the smart card industry.

After more than 20 years of standardization effort, the most important basic ISO standards for smart cards are now complete. They form the basis for further, application-specific standards, which are currently being prepared by ISO and CEN. These standards are based on prior ISO standards in the 7810, 7811, 7812 and 7813 families, which define the properties of identification cards in the ID-1 format. These standards include embossed cards and cards with magnetic stripes, which we all know in the form of credit cards.

Compatibility with these existing standards was a criterion from the very beginning in the development of standards for smart cards (which are called 'integrated circuit(s) cards', or 'ICC', in the ISO standards), in order to provide a smooth transition from embossed cards and magnetic-stripe cards to smart cards. Such a transition is possible because all functional components, such as embossing, magnetic stripes, contacts and interface components for contactless interfaces, can be integrated into a single card. Of course, a consequence of this is that the integrated circuits, which are sensitive electronic components, are exposed to high stresses during the embossing process and recurrent impact stresses when the embossed characters are printed onto paper. This makes heavy demands on the packaging of the integrated circuits and the manner in which they are embedded in the card.

A summary of the currently available standards, with brief descriptions of their contents, can be found in the Appendix.¹

In the last few years, an increasing number of specifications have been prepared and published by industrial organizations and other non-public groups, with no attempt being made to incorporate them into the standardization activities of ISO. The argument most commonly offered for this manner of working is that the way ISO operates is too slow to keep pace with the short innovation cycles of the informatics and telecommunication industries. Since

¹ See Section 16.4, 'Annotated Directory of Standards and Specifications'

frequently only a few companies are involved in drafting these ‘industry standards’, there is a large risk that the interests of smaller companies, and especially the interests of the general public, will be ignored in the process. It is a major challenge to the future of ISO to devise a working method that can safeguard general interests without hampering the pace of innovation.

2

Types of Cards

As already mentioned in the Introduction, smart cards are the youngest member of the family of identification cards using the ID-1 format defined in ISO standard 7810, 'Identification Cards – Physical Characteristics'. This standard specifies the physical properties of identification cards, such as flexibility and temperature resistance, as well as the dimensions of three different card formats: ID-1, ID-2 and ID-3. The smart card standards (ISO 7816-1 ff) are based on ID-1 cards, millions of which are used nowadays for financial transactions.

This chapter provides an overview of various types of cards in the ID-1 format, since a combination of various functions is of particular interest for many applications, especially when the cards currently used in an existing system (such as magnetic-stripe cards) are to be replaced by smart cards. In such cases, it is usually not possible to replace the existing infrastructure (such as magnetic-stripe card terminals) by a new technology overnight.

The solution to this problem generally consists of issuing cards with both magnetic stripes and chips, for use during a transition period. Such cards can be used with both types of terminals (old and new). Naturally, new functions that are only possible with a chip cannot be used with a terminal that only supports magnetic-stripe cards.

2.1 EMBOSSED CARDS

Embossing is the oldest technique for adding machine-readable features to identification cards. The embossed characters on the card can be transferred to paper using simple, inexpensive devices, and they can also be easily read visually (by humans). The nature and location of the embossing are specified in the ISO 7811 standard ('Identification Cards – Recording Technique'). This standard, which is divided into five parts, deals with magnetic stripes as well as embossing.

ISO 7811 Part 1 specifies the requirements for embossed characters, including their form, size and embossing height. Part 3 defines the precise positioning of the characters on the card and defines two separate regions, as shown in Figure 2.1. Region 1 is reserved for the card's identification number, which identifies the card issuer as well as cardholder. Region 2 is reserved for additional data relating to the cardholder, such as his or her name and address.

At first glance, transferring information by printing from embossed characters may appear quite primitive. However, the simplicity of this technique has made worldwide proliferation of credit cards possible, even in developing countries. The exploitation of this technology requires neither electrical energy nor a connection to a telephone network.

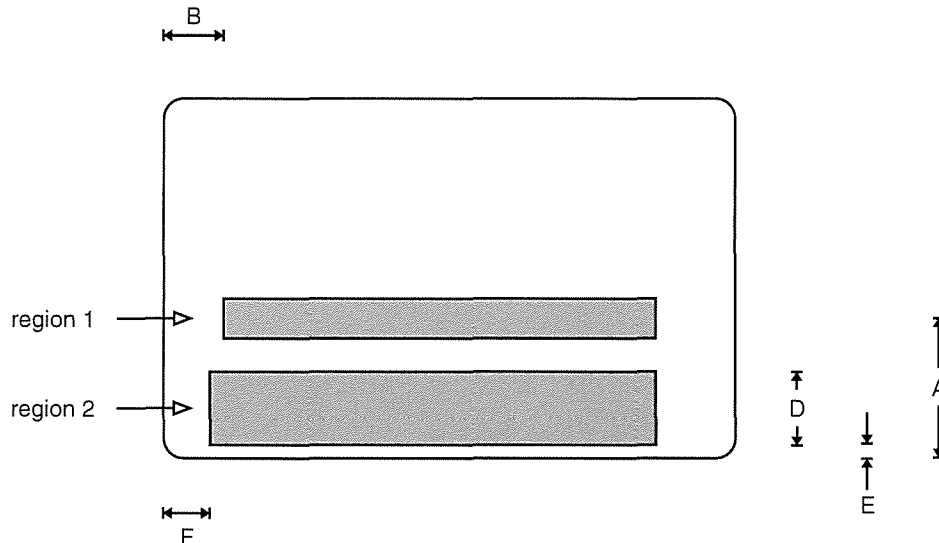


Figure 2.1 Embossing locations according to ISO 7811-3. Region 1 is reserved for the ID number (19 characters), and region 2 is reserved for the cardholder's name and address (4×27 characters). $A = 21.42 \pm 0.12$ mm, $B = 10.18 \pm 0.25$ mm, $D = 14.53$ mm, $E = 2.41 - 3.30$ mm, $F = 7.65 \pm 0.25$ mm

2.2 MAGNETIC-STRIPE CARDS

The fundamental disadvantage of embossed cards is that their use creates a flood of paper receipts, which are expensive to process. One remedy for this problem is to digitally encode the card data on a magnetic stripe located on the back of the card. The magnetic stripe is read by pulling it across a read head, either manually or automatically, with the data being read and stored electronically. No paper is required to process the data.

Parts 2, 4 and 5 of ISO standard 7811 specify the properties of the magnetic stripe, the coding technique and the locations of the magnetic tracks. The magnetic stripe may contain up to three tracks. Tracks 1 and 2 are specified to be read-only tracks, while track 3 may also be written to.

Although the storage capacity of the magnetic stripe is only about 1000 bits, which is not very much, it is nevertheless more than sufficient for storing the information contained in the embossing. Additional data can be read and written on track 3, such as the most recent transaction data in the case of a credit card.

The main drawback of magnetic-stripe technology is that the stored data can be altered very easily. Manipulating embossed characters requires at least a certain amount of manual dexterity, and such manipulations be easily detected by a trained eye. By contrast, the data recorded on the magnetic stripe can be altered relatively easily using a standard read/write device, and it is difficult to afterwards prove that the data have been altered. Furthermore,

magnetic-stripe cards are often used in automated equipment in which visual inspection is not possible, such as cash dispensers. A potential criminal, having obtained valid card data, can easily use duplicated cards in such unattended machines without having to forge the visual security features of the cards.

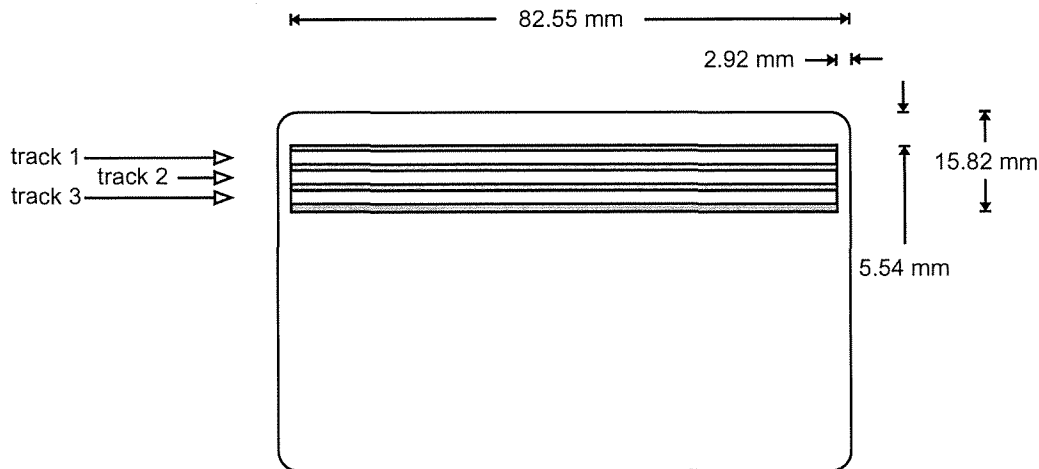


Figure 2.2 Location of the magnetic stripe on an ID-1 card. The data region of the magnetic stripe is intentionally not extended to the edges of the card, since the use of hand-operated card readers causes rapid wear at the ends of the stripe

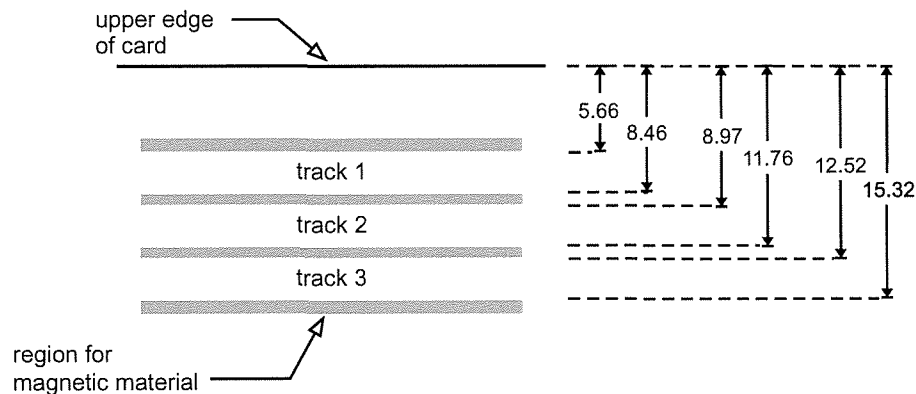


Figure 2.3 Locations of the individual tracks on an ID-1 card (all dimensions in mm)

Manufacturers of magnetic-stripe cards have developed various means to protect the data recorded on the magnetic stripe against forgery and duplication. For example, German Eurocheque cards contain an invisible, unalterable code in the body of the card, which effectively makes it impossible to alter or duplicate the data on the magnetic stripe. However, such techniques require a special sensor in the card terminal, which considerably increases the cost of the terminal. For this reason, none of these techniques has so far succeeded in becoming internationally established.

Table 2.1 Standard features of the three tracks on a magnetic-stripe card, as specified in ISO 7811

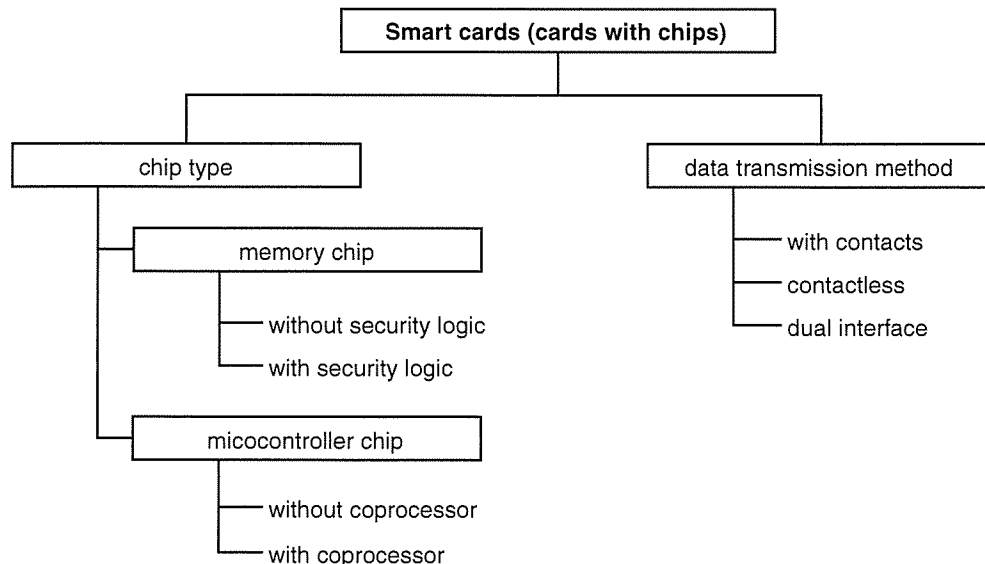
Feature	Track 1	Track 2	Track 3
Amount of data	79 characters max	40 characters max	107 characters max
Data coding	6-bit alphanumeric	4-bit BCD	4-bit BCD
Data density	210 bpi (8.3 bit/mm)	75 bpi (3 bit/mm)	210 bpi (8.3 bit/mm)
Writing	not allowed	not allowed	allowed

2.3 SMART CARDS

The smart card is the youngest and cleverest member of the family of identification cards in the ID-1 format. Its characteristic feature is an integrated circuit embedded in the card, which has components for transmitting, storing and processing data. The data can be transmitted using either contacts on the surface of the card or electromagnetic fields, without any contacts.

Smart cards offer several advantages compared with magnetic-stripe cards. For instance, the maximum storage capacity of a smart card is many times greater than that of a magnetic-stripe card. Chips with more than 256 kB of memory are currently available, and this figure will multiply with each new chip generation. Only optical memory cards, which are described in the next section, have greater capacities.

However, one of the most important advantages of smart cards is that their stored data can be protected against unauthorized access and manipulation. Since the data can only be accessed via a serial interface that is controlled by an operating system and security logic, confidential data can be written to the card and stored in a manner that prevents them from ever being read from outside the card. Such confidential data can be processed only internally by the chip's

**Figure 2.4** Classification chart for cards containing chips according to the type of chip used and the method used for data transmission

processing unit. In principle, both hardware and software mechanisms can be used to restrict the use of the storage functions of writing, erasing and reading data and tie them to specific conditions. This makes it possible to construct a variety of security mechanisms, which can also be tailored to the specific requirements of a particular application. In combination with the ability to compute cryptographic algorithms, this allows smart cards to be used to implement convenient security modules that can be carried by users at all times, for example in a purse or wallet. Some additional advantages of smart cards are their high level of reliability and long life compared with magnetic-stripe cards, whose useful life is generally limited to one or two years.

The fundamental characteristics and functions of smart cards are specified in the ISO 7816 family of standards, which are described in detail in the following chapters.

Smart cards can be divided into two groups, which differ in both functionality and price: *memory cards* and *microprocessor cards*.

2.3.1 Memory cards

Figures 2.5 and 2.6 show architectural block diagrams of memory cards.

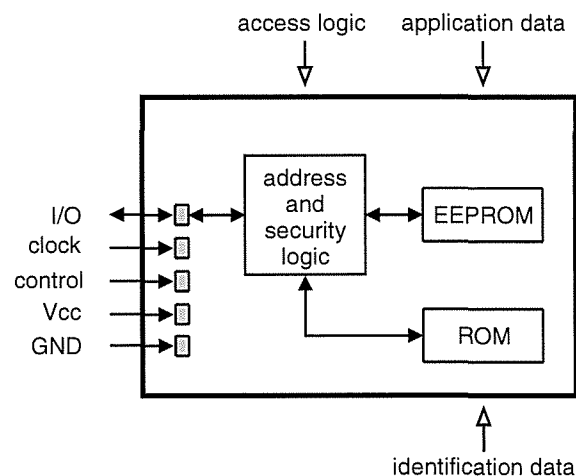


Figure 2.5 Typical architecture of a contact-type memory card with security logic. The figure shows only basic energy and data flows and is not a detailed schematic diagram

The data needed by the application are stored in the memory, which is usually EEPROM. Access to the memory is controlled by the security logic, which in the simplest case consists only of write protection or erase protection for the memory or certain memory regions. However, there are also memory chips with more complex security logic that can also perform simple encryption. Data are transferred to and from the card via the I/O port. Part 3 of the ISO 7816 standard defines a special synchronous transfer protocol that allows the chip implementation to be particularly simple and inexpensive. However, some smart cards use the I²C bus, which is commonly used for serial-access memories.

The functionality of memory cards is usually optimized for a particular application. Although this severely restricts the flexibility of the cards, it makes them quite inexpensive. Memory cards are typically used for prepaid telephone cards and health insurance cards.

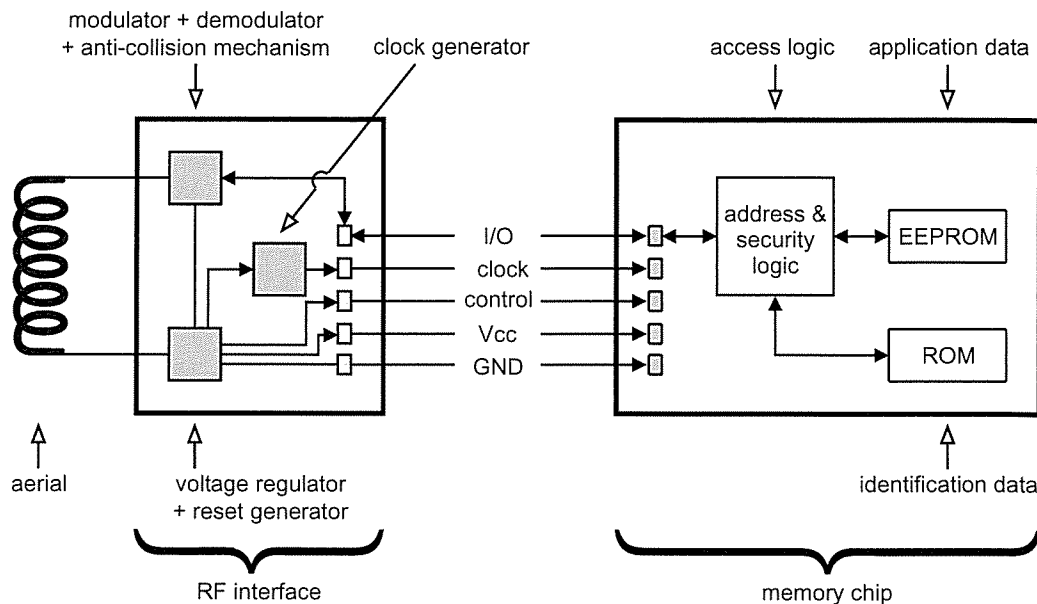


Figure 2.6 Typical architecture of a memory card with security logic and a contactless interface. The figure shows only basic energy and data flows and is not a detailed schematic diagram

2.3.2 Microprocessor cards

The heart of the chip in a microprocessor card, as the name suggests, is a processor, which is usually surrounded by four additional functional blocks: mask ROM, EEPROM, RAM and an I/O port. Figure 2.7 shows the architecture of a typical device of this type.

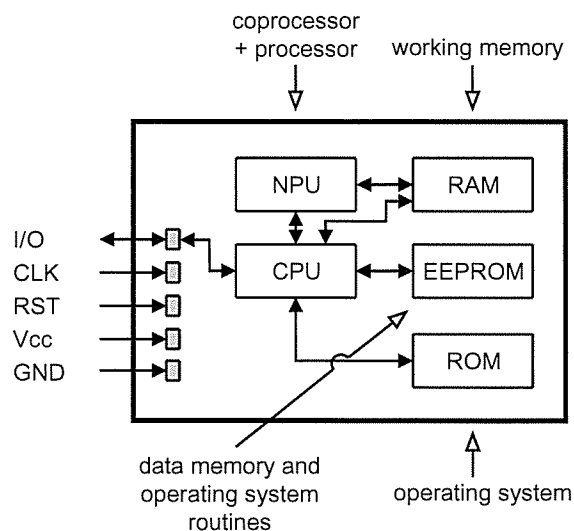


Figure 2.7 Typical architecture of a contact-type microprocessor card with a coprocessor. The figure shows only basic energy and data flows and is not a detailed schematic diagram

The mask ROM contains the chip's operating system, which is 'burned in' when the chip is manufactured. The content of the ROM is thus identical for all the chips of a production run, and it cannot be changed during the chip's lifetime. The EEPROM is the chip's non-volatile memory. Data and program code can be written to and read from the EEPROM under the control of the operating system. The RAM is the processor's working memory. This memory is volatile, so all the data stored in it are lost when the chip's power is switched off. The serial I/O interface usually consists only of a single register, via which data are transferred bit by bit.

Microprocessor cards are very flexible in use. In the simplest case, they contain a program optimized for a single application, so they can only be used for this particular application. However, modern smart card operating systems allow several different applications to be integrated into a single card. In this case, the ROM contains only the basic components of the operating system, with the application-specific part of the operating system being loaded into the EEPROM only after the card has been manufactured. Recent developments even allow application programs to be loaded into a card after it has already been personalized and issued to the cardholder. Special hardware and software measures are used to prevent the security conditions of the individual applications from being violated by this capability. Special microprocessor chips with high processing capacities and large memory capacities, which are optimized for such use, are now available.

2.3.3 Contactless smart cards

Electrical connections with contact-type smart cards are made via the eight contacts specified in the ISO 7816 Part 1 standard. The reliability of contact-type smart cards has been steadily improved over the past years as the result of experience accumulated in manufacturing such cards. The failure rate of telephone cards within their one-year service life, for instance, is currently significantly less than one in a thousand. Nevertheless, contacts are one of the most frequent sources of failure in electromechanical systems. Disturbances can be caused by factors such as contamination and contact wear. In mobile equipment, vibrations can cause brief intermittent contacts. Since the contacts on the surface of the card are directly connected to the inputs of the integrated circuit chip embedded in card, there is a risk that the chip may be damaged or destroyed by electrostatic discharge. Static charges of several thousand volts are by no means rare.

These technical problems are elegantly avoided by contactless smart cards. In addition to its technical advantages, contactless-card technology offers card issuers and cardholders a range of new and attractive potential applications. For instance, contactless cards do not necessarily have to be inserted into a card reader, since there are systems available that work at a range of up to one meter. This is a great advantage in access-control systems where a door or turnstile has to be opened, since the access authorization of a person can be checked without requiring the card to be removed from a purse or pocket and inserted into a reader. One major application area for this technology is local public transport, which requires a large number of people to be identified in the shortest possible time.

However, contactless technology is also advantageous in systems that do require deliberate insertion of the card into a reader, since it does not matter how the card is inserted in the reader. This contrasts with magnetic cards or cards with contacts, which work only with a specific card

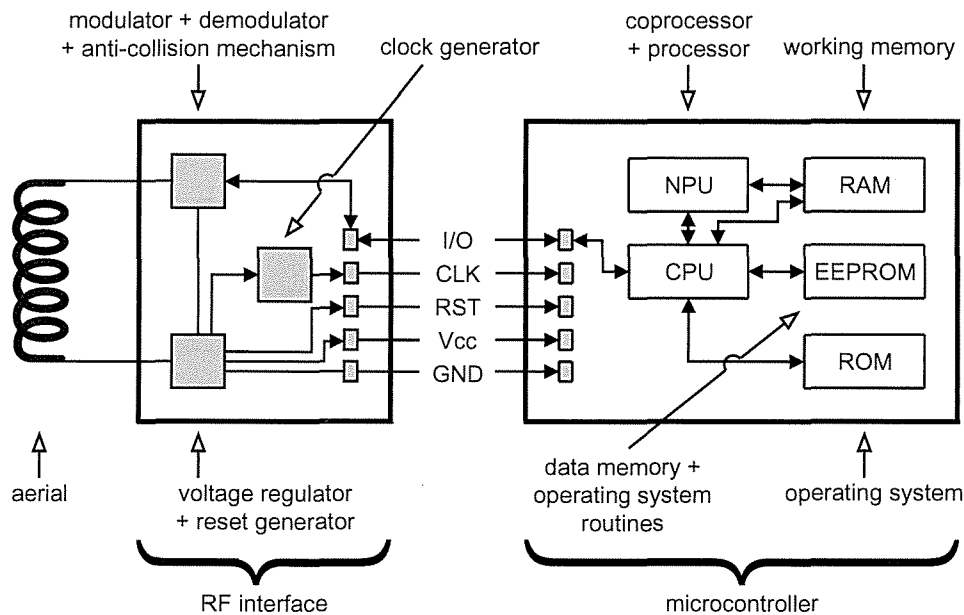


Figure 2.8 Typical architecture of a microprocessor card with a coprocessor and a contactless interface. The figure shows only basic energy and data flows and is not a detailed schematic diagram

orientation. Freedom from orientation restrictions simplifies use and thus increases customer acceptance.

A further interesting variation on using contactless cards involves a 'surface terminal'. In this case, the card is not inserted into a slot, but simply placed on a marked location on the surface of the card reader. In addition to simplicity of use, this solution is attractive because it significantly reduces the risk of vandalism (for example, forcing chewing gum or superglue into the card slot).

For card marketing, contactless technology offers the advantage that no technical components are visible on the card surface, so visual design is not constrained by magnetic stripes or contacts. However, this advantage comes at the price of more complex terminals with correspondingly higher prices. Another disadvantage is that several different systems for contactless smart cards have been standardized and marketed, further increasing the complexity of terminals that must be compatible with all standardized cards.

Manufacturing technology for the mass production of contactless cards has matured to the point that high-quality products are available at prices that do not significantly differ from those of comparable contact-type cards. Up to now, contactless cards have predominantly been used in local public transportation systems, in which they serve as electronic tickets in modern electronic-fare systems. Most of the systems presently in use employ single-function cards containing inexpensive chips with hard-wired security logic. However, there is a growing demand for incorporating value-added services in electronic tickets. Multifunction cards with integrated microprocessors are thus being used increasingly often, with the payment function most commonly being implemented using the conventional contact-based technique in order to utilize existing infrastructures, such as electronic purse systems. These new multifunction

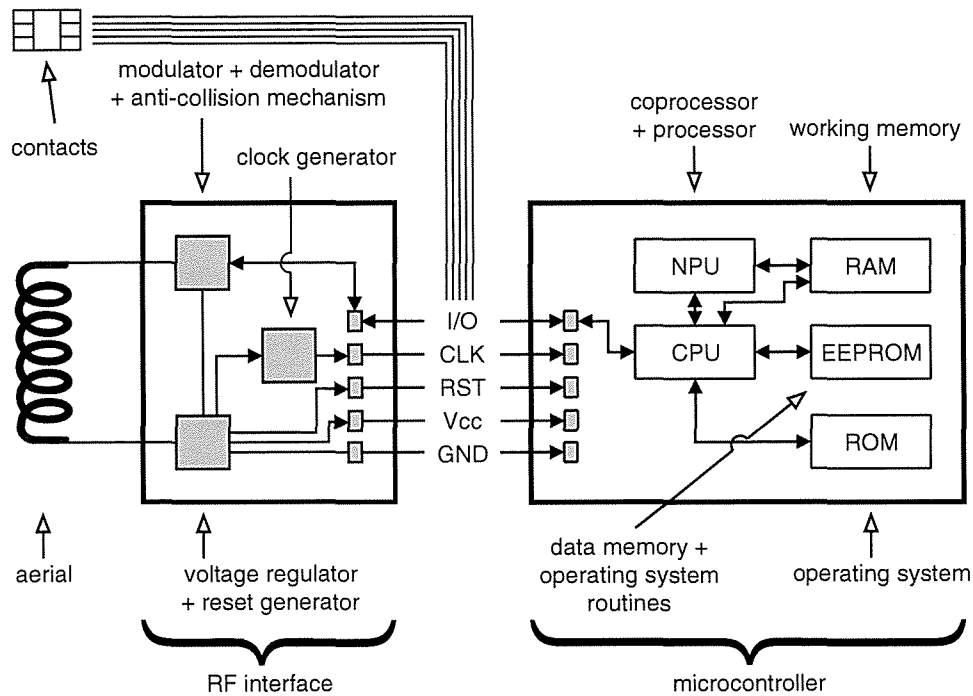


Figure 2.9 Typical architecture of a microprocessor card with a coprocessor and both contactless and contact interfaces. The figure shows only basic energy and data flows and is not a detailed schematic diagram

cards have both contact and contactless coupling elements and are called 'dual-interface cards' or 'combicards'.

The technology and operating principles of contactless smart cards are described in detail in Section 3.6, 'Contactless Smart Cards'.

2.4 OPTICAL MEMORY CARDS

For applications where the storage capacity of smart cards is insufficient, optical cards that can store several megabytes of data are available. However, with current technology these cards can be written only once and cannot be erased.

The ISO/IEC 11 693 and 11 694 standards define the physical characteristics of optical memory cards and the linear data-recording technique used with such cards.

Combining the large storage capacity of optical memory cards with the intelligence of smart cards leads to interesting new possibilities. For example, data can be written in encrypted form to the optical memory, with the key being securely stored in the private memory of the chip. This protects the optically stored data against unauthorized access.

Figure 2.11 shows the typical layout of an optical smart card with contacts, a magnetic stripe and an optical storage region. It can be seen that the area available for optical storage

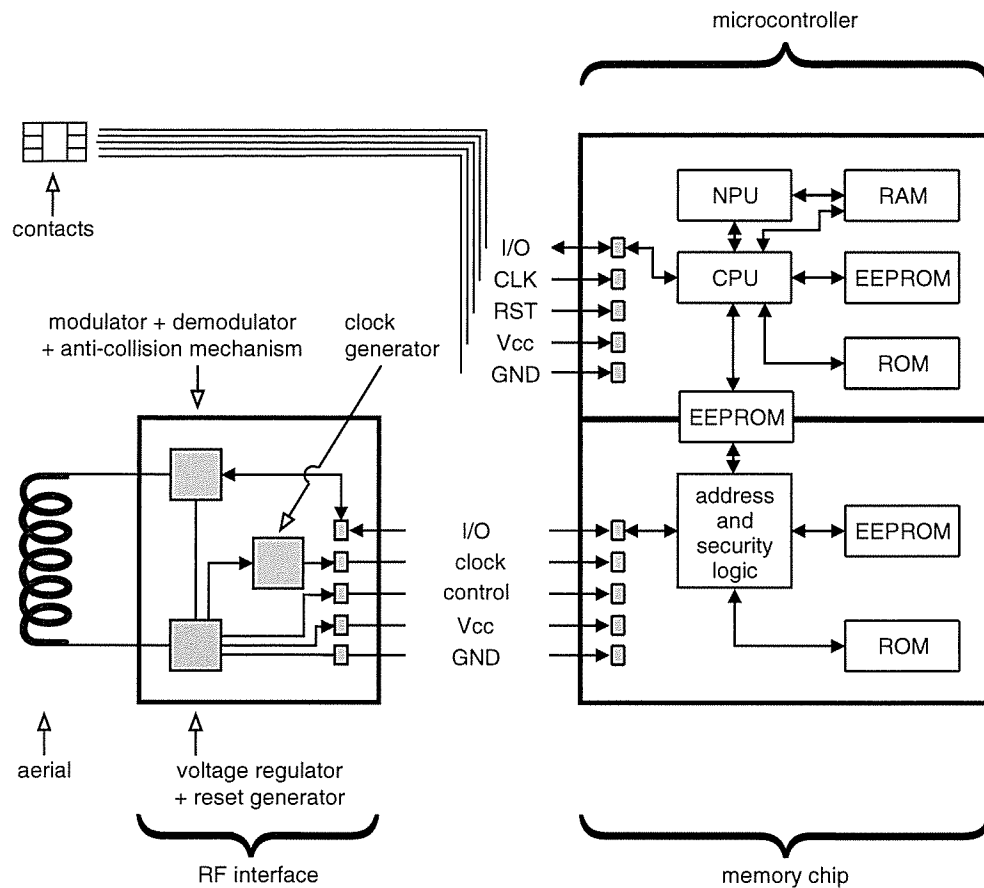


Figure 2.10 Typical architecture of a dual-interface card, which is a combination of a contactless memory card and a contact-type microprocessor card. The figure shows only basic energy and data flows and is not a detailed schematic diagram

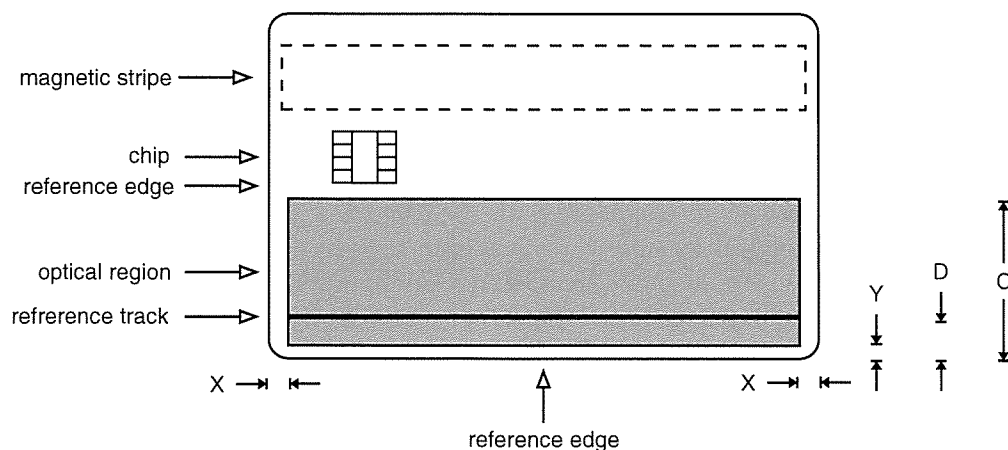


Figure 2.11 Location of the optical storage area on an ID-1 card according to ISO/IEC 11 694-2. $C = 9.5\text{--}49.2\text{ mm}$, $D = 5.8 \pm 0.7\text{ mm}$, $X = 3\text{ mm max with PWM or }1\text{ mm max with PPM}$, $Y = 1\text{ mm min with PWM } (Y < D) \text{ or } 4.5\text{ mm max with PPM}$ (PWM = pulse-width modulation, PPM = pulse-position modulation)

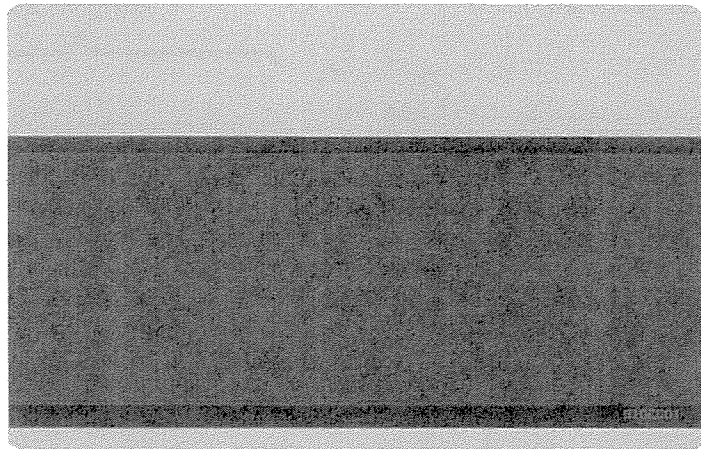


Figure 2.12 A typical optical memory card with a net storage capacity (with error correction) of approximately 4 MB. The raw capacity (without error correction) is approximately 6 MB

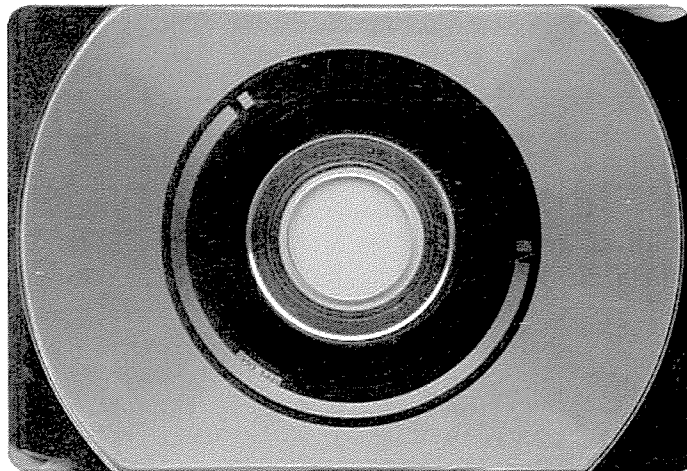


Figure 2.13 An optical memory card with a storage capacity of approximately 32 MB that can be read by a CD-ROM drive

is limited by the contacts for the chip, which naturally reduces the total storage capacity. The magnetic stripe is located on the rear of the card.

Up to now, use of optical memory cards has been severely limited by the high cost of equipment for reading and writing this type of card. One application for optical memory cards is recording patient data in the medical sector, since their large storage capacity allows even X-ray images to be stored on a card.

3

Physical and Electrical Properties

The card body of a smart card inherits its fundamental properties from its predecessor, the familiar embossed card, which still dominates the market in the credit card sector. Technically speaking, such cards are simple plastic structures that are personalized by being embossed with a variety of user features, such as the name and number of the cardholder. Later versions of these cards were provided with a magnetic stripe to enable simple machine processing. When the idea of implanting a chip in the card first arose, this existing type of card was used as the basis and a microcontroller was embedded in the body of the card. Many standards relating to the card's physical properties are thus not specific to smart cards, but apply equally well to magnetic-stripe and embossed cards.

3.1 PHYSICAL PROPERTIES

If you hold a smart card in your hand, the first thing you notice is its format. After this, you might see that it has set of contacts, although a contactless smart card may not have any visible electrical interface. The next feature to catch your eye might be a magnetic stripe, embossing or a hologram. All these features and functional components form part of the physical characteristics of a smart card.

Most of the physical characteristics are actually purely mechanical in nature, such as the format of the card and its resistance to bending or twisting. These characteristics are familiar to every user from personal experience. In practice, however, physical properties such as sensitivity to temperature or light and resistance to moisture are also important.

The interaction between the body of the card and the implanted chip must also be considered, since only the combination of the two components makes a functional card. For instance, a card body designed for use at high ambient temperatures is of little benefit if its embedded microcontroller does not share this property. These two components must individually and collectively meet all of the relevant requirements, since otherwise high failure rates can be expected in use.

3.1.1 Card formats

Small cards with the typical smart card dimensions of 85.6 mm by 54 mm have been in use for a very long time. Almost all smart cards are produced in this format, which is certainly the most familiar. It is designated ID-1, and its size is specified in the ISO 7810 standard. This standard originated in 1985 and thus has nothing to do with smart cards as we know them today, as can easily be seen from the abbreviation 'ID', which stands for 'identification'. This standard simply describes an embossed plastic card with a magnetic stripe that is intended to be used for the identification of a person. When it was written, no one had thought of putting a chip into the card. The presence of a chip and location of its contacts on the card were only defined several years later in other standards.

With the variety of cards available today, which are used for all possible purposes and have a wide range of dimensions, it is often difficult to determine whether a particular card is actually an ID-1 smart card. In addition to the embedded chip, one of the best identifying features is the thickness of the card. If this measures 0.76 mm and the card contains a microcontroller, it can be considered to be a smart card in the sense of the ISO standard.

The conventional ID-1 format has the advantage of being very easy to handle. The card's format is specified such that it is not too large to be carried in a wallet, but not so small that it is easily lost. In addition, the card's flexibility makes it more convenient than a rigid object.

Nevertheless, this format does not always meet the demands of modern miniaturization. Some mobile telephones weigh only 200 g and are not much bigger than a packet of tissues. It thus became necessary to define a smaller format in addition to the ID-1 format, in order to address the needs of small terminal devices. The card used in such devices can be very small, since it is usually inserted into the device only once and remains there for good. The ID-000 format was defined for this purpose, and it bears the descriptive name 'plug-in'. This format is presently only used with GSM mobile telephones, which have very little room for a card and do not require the card to be frequently exchanged.

However, the fact that cards in the ID-000 format are inconvenient to handle, both in production and by end users, led to the development of an additional format. This format is designated ID-00, or 'mini-card'. Its dimensions are approximately halfway between those of ID-1 and ID-000 cards. This type of card is more convenient to handle and cheaper to produce, for instance because it is easier to print. However, the ID-00 definition is fairly new, and this format has not yet become established either nationally or internationally.

The formats are defined in the relevant standards in a way that simplifies measuring the card dimensions. For instance, the height and width of an ID-1 card must be such that it fits between two concentric rectangles as shown in Figure 3.1 (ignoring the rounded corners), which have the following dimensions:

- external rectangle: width: 85.72 mm (3.375 inches)
 height: 54.03 mm (2.127 inches)
- internal rectangle: width: 85.46 mm (3.365 inches)
 height: 53.92 mm (2.123 inches)

The thickness must be 0.76 mm (0.03 inch), with a tolerance of ± 0.08 mm (± 0.003 inch). The corner radii and the card's thickness are defined conventionally. Based on these definitions, an ID-1 card's dimensions can be represented as shown in Figure 3.2.

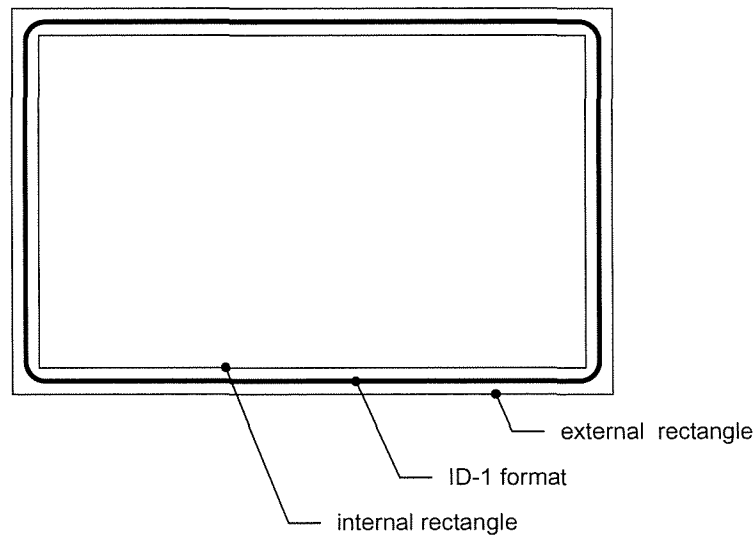


Figure 3.1 Definition of the dimensions of an ID-1 format card

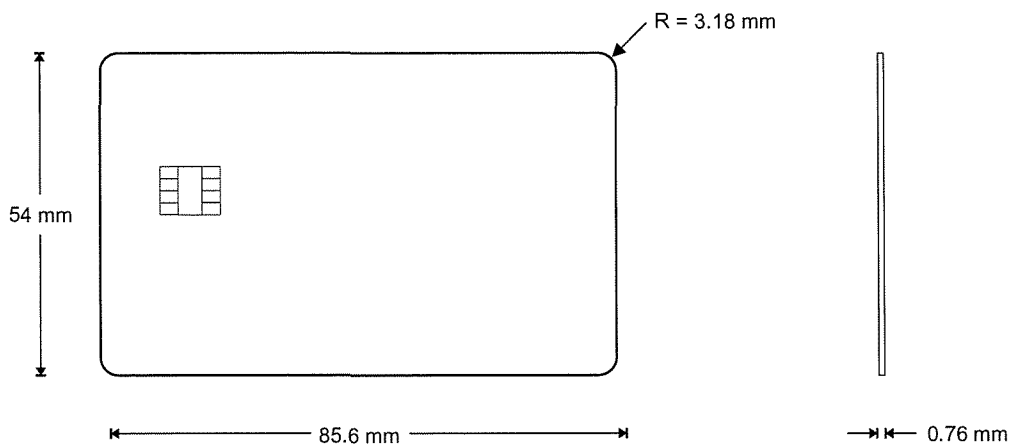


Figure 3.2 The ID-1 format. Thickness: $0.76 \text{ mm} \pm 0.08 \text{ mm}$; corner radius: $3.18 \text{ mm} \pm 0.30 \text{ mm}$. The dimensions shown indicate the size of the card excluding tolerances

The ID-000 format is also defined using two concentric rectangles. Since this format originated in Europe (based on the GSM mobile telephone system), the basic dimensions are metric. The bottom right-hand corner of a plug-in card is cut off at an angle of 45° , as shown in Figure 3.3, in order to facilitate correct insertion of the card into the card reader.

The dimensions of the two rectangles for the ID-000 format are:

- external rectangle: width: 25.10 mm
 height: 15.10 mm
- internal rectangle: width: 24.90 mm
 height: 14.90 mm

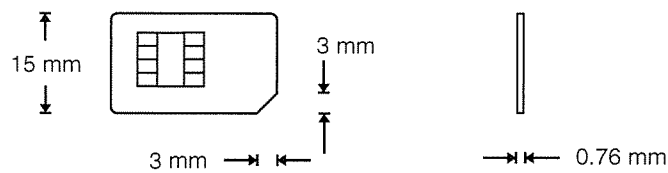


Figure 3.3 The ID-000 format. Thickness: $0.76 \text{ mm} \pm 0.08 \text{ mm}$; corner radius: $1 \text{ mm} \pm 0.10 \text{ mm}$; corner: $3 \text{ mm} \pm 0.03 \text{ mm}$. The dimensions shown indicate the size of the card excluding tolerances

The ID-00 format is also based on metric measurements. Its maximum and minimum dimensions are defined by two concentric rectangles with the following dimensions:

- external rectangle: width: 66.10 mm
 height: 33.10 mm
- internal rectangle: width: 65.90 mm
 height: 32.90 mm

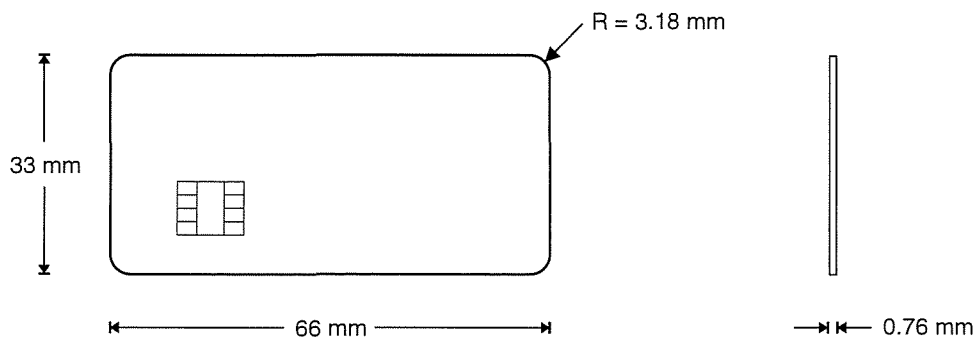


Figure 3.4 The ID-00 format. Thickness: $0.76 \text{ mm} \pm 0.08 \text{ mm}$; corner radius: $3.18 \text{ mm} \pm 0.30 \text{ mm}$. The dimensions shown indicate the size of the card excluding tolerances

The relative sizes of the ID-1, ID-00 and ID-000 formats are shown in Figure 3.5. Cards in the smaller formats can be produced from the larger versions by punching them from the body of a larger-format card. This is especially important for card manufacturers, since it allows the production process to be optimized and made more economical using a uniform ID-1 format. For instance, card manufacturers commonly produce card blanks in only one format (preferably ID-1), embed modules in them and fully personalize them. Depending on the specific application of the cards so produced, such cards can be trimmed to the desired format in a subsequent production step.

Alternatively, the format may be modified later by the customer. This has become common practice with cards for mobile telephones. The customer receives an ID-1 card that is prepunched such that it can be converted into an ID-000 card by breaking a small card free from the larger card body. In another technique, the ID-000 card is completely punched free from the ID-1 body and attached to the surrounding portion using single-sided tape on the side without contacts. The customer can thus 'produce' a card whose format fits his or her equipment, while the manufacturer only has to produce and supply one card format.

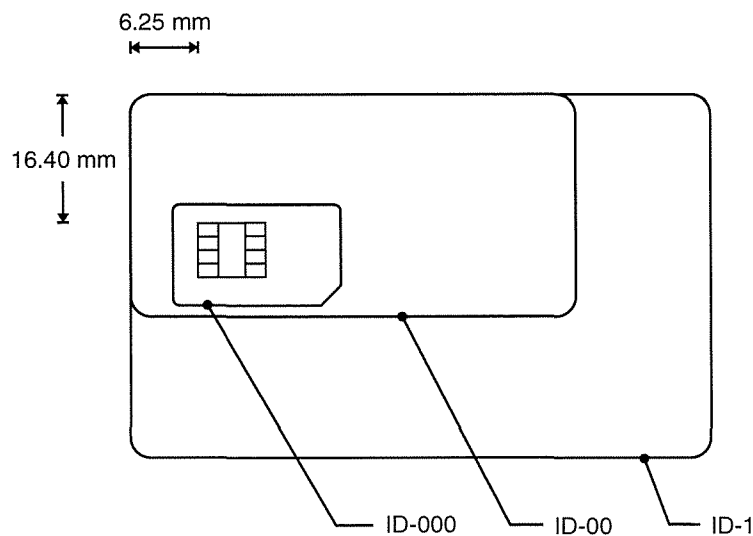


Figure 3.5 Relative sizes of the ID-1, ID-00 and ID-000 formats

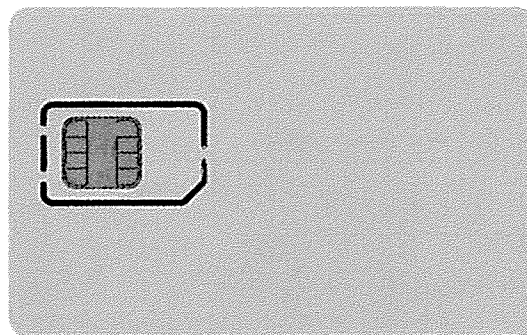


Figure 3.6 Example of a mobile telephone card in ID-1 format, which the user can convert into an ID-000 card if necessary by pressing out the smaller-format card

However, the usual card format still has some disadvantages for some applications. In such situations, other form factors can be used, such as a USB plug with an integrated smart card microcontroller. The logical behavior of such smart card variants is generally fully equivalent to that of the usual forms.

3.1.2 Card components and security features

Since smart cards are primarily used to provide authorization for specific actions or identify cardholders, security features on the card body are often needed in addition to the embedded chip. Since the authenticity of the card may be verified by humans as well as by machines, many security features are based on visual features. However, some security features employ a modified smart card microcontroller and thus can only be verified by a computer. In contrast to the security features used with microcontrollers, the usual features for human verification of

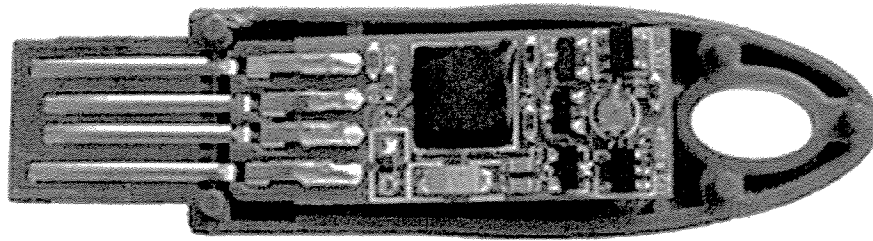


Figure 3.7 Example of a smart card with a different form factor. This photo shows a USB plug with a soldered-in smart card microcontroller and the necessary interface components, which has been opened up to reveal its internal components

the authenticity of a card are not based on cryptographic procedures (such as mutual authentication). Instead, they are primarily based on using secret materials and production processes or using processes whose mastery requires a large amount of effort or considerable expertise, or that are technically difficult.

Particularly in the area of new card components, there is considerable potential for new developments in the near future with regard to the integration of additional components such as keypads, displays, solar cells and batteries.

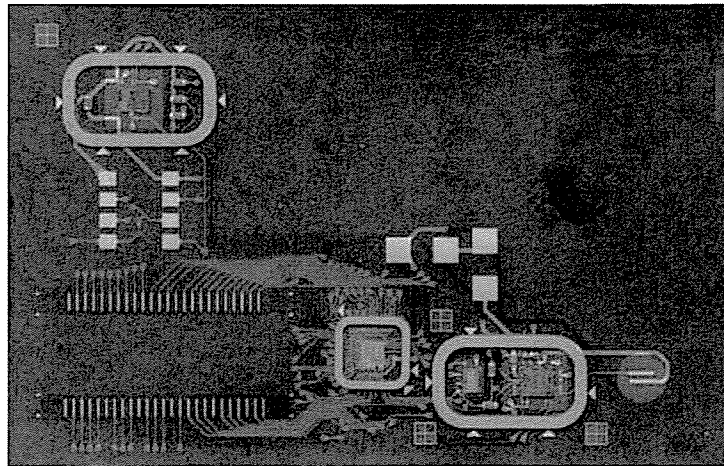


Figure 3.8 Inlay foil for a super smart card. The actual smart card microcontroller can be seen at the left, connected to the contact below it. The contact pads for a display are located at the lower left. The driver IC for the display is located to the right of the contact pads, with four large contacts for the battery above it. The contacts for a pushbutton switch can be seen at the lower right, with various components for the interface adapter located to the left (*Source: Giesecke & Devrient*)

Signature panels

A very simple way to identify the cardholder is to use a signature panel attached to the card, as is common with credit cards. Once such a panel has been signed, it cannot be altered, so it

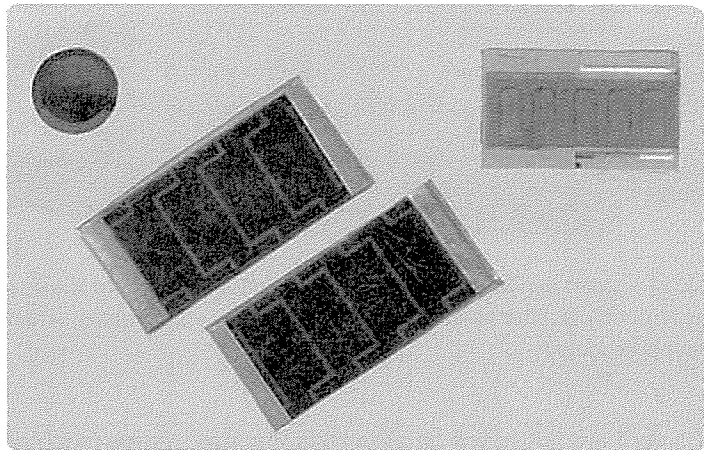


Figure 3.9 Early laboratory prototype of a super smart card for an electronic purse system using contactless cards. A pushbutton switch for confirming transactions can be seen at the upper left, with two solar cells in the middle to supply power. A five-digit display for showing the purse balance and other data is located at the upper right (*Source: Giesecke & Devrient*)

is erasure-proof. A very fine colored pattern printed on the panel makes any attempt to cover the panel immediately apparent. The signature panel is permanently bonded to the card body by using a hot-gluing process to attach a printed paper strip to the card. Alternatively, the signature panel may be part of the top layer of the card, which is laminated into the card when it is assembled.

Guilloche patterns

A somewhat more complicated technique is to place a foil printed with guilloche patterns under the transparent outer layer of the card. Guilloche patterns are decorative patterns consisting of very fine interwoven lines, usually round or oval, such as are found on some bank notes and share certificates. These patterns have such fine structures that they can presently only be produced by printing processes, and are thus difficult to copy.

Microtext

Another technique that is based on the security provided by fine printed line structures is using microtext lines. These appear to be plain lines to the naked eye, but they can be recognized as text using a loupe. Like guilloche patterns, microtext cannot be photocopied.

Ultraviolet text

In order not to affect the visible layout of the card, control characters or control numbers can be printed on the card using ink that is only visible under ultraviolet light. However, this technique provides only relatively limited protection against forgery.

Barcodes

For storing a small amount of data, a barcode can be printed on the surface of the card using laser engraving or thermal-transfer printing. The advantage of barcodes is that they can be automatically read at close range using optical equipment. The barcodes used on smart cards include not only the widely used one-dimensional type, but also two-dimensional barcodes in the form of stacked or matrix barcodes. A two-dimensional matrix barcode, such as PDF 417 for example, can easily encode up to 1000 bytes, and if an integrated Reed–Solomon code is used for error correction, the data can be recovered even when up to 25 percent of the barcode area is unreadable.

Holograms

A hologram integrated into the card is a security feature that by now is familiar to all card users. The security of holograms is primarily based on the fact that they are produced by only a few companies in the world and that they are not readily available.

The holograms used for smart cards are called ‘embossed’ holograms. Since they can be viewed using diffuse reflected daylight, they are also referred to as ‘white-light reflection holograms’. By contrast, a conventional transmission hologram must be viewed using coherent laser light. Supplementary security features that can only be seen with laser light are sometimes integrated into the hologram as well. In order to produce an embossed hologram, it is necessary to first generate a master hologram using the conventional holographic technique. A master embossing stamp is then prepared from the master hologram using a transfer process. The embossing stamp contains the microstructures that will produce the subsequent embossed holograms. Daughter stamps are prepared from the master stamp using electroplating processes, and these daughter stamps are used to emboss the hologram structure in plastic films. These films are then coated with a layer of vaporized aluminum to produce the well-known white-light reflection holograms.

The hologram is permanently bonded to the card body, so it cannot be removed without destroying it. This can be done using either a lamination process or the ‘roll-on’ process. In the latter process, a hologram located on a carrier film is pressed onto the card by a heated roller. The carrier film is then pulled off, and the hologram remains permanently welded to the plastic card body. A third process that is used is the ‘hot-stamping’ process, which is similar to the roll-on process except that a heated stamp is used instead of a heated roller.

Kinegrams

Kinegrams, which are popularly called ‘3-D images’, are made in the same way as holograms. The viewer sees an image that changes abruptly when the viewing angle is changed. Kinegrams are just as hard to forge as holograms, and they have the advantage that they are more quickly recognized by the viewer and thus can be verified more quickly.

Multiple laser image (MLI)

A multiple laser image is a sort of kinegram that is very similar to a simple hologram. It uses an array of lenses pressed into the surface of the card, some of which have been blackened by

a laser. The main difference between an MLI and a hologram is that card-specific information is shown in the small MLI image. For instance, this technique can be used to mark the name of the cardholder on an individual card in the form of a kinegram.

Laser engraving

Darkening a special plastic layer by heating it with a laser beam is called laser engraving, or simply 'lasing'. In contrast to embossing, this is a secure way to write data on an individual card, such as the cardholder's name and the card number. It is secure because the necessary equipment and the knowledge of how to use it are not readily available.

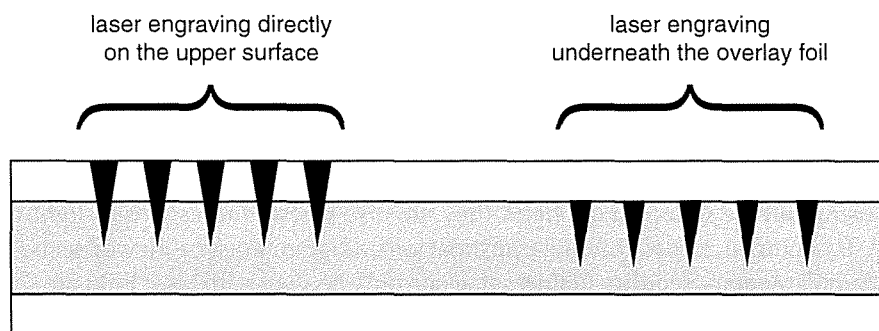


Figure 3.10 Cross-section of laser engraving in a card (not to scale). Laser engraving can take place either on the surface of the card or in an internal layer below a cover foil that is transparent to the laser light

Two different methods are used for laser engraving: vector engraving and raster engraving. In the vector method, the laser beam is directed along its path without interruption. This is very well suited to writing characters and has the advantage of being quick. In the raster technique, by contrast, a large number of adjacent points are blackened to produce an image, similar to the operation of an ink-jet or dot-matrix printer. This method is primarily used to place a picture on the card. Although it has the advantage of high resolution, which allows details to be reproduced well, it has the disadvantage of being very time-consuming. For instance, it takes approximately 10 seconds to laser-engrave a standard-quality passport photograph.

Embossing

Another way to add user data to a card is to emboss characters onto the card. This is done by hammering metal letter punches against the card. In principle, this process works the same way as a mechanical typewriter. Nowadays, the only benefit of embossing is that the embossed characters can easily be transferred to preprinted forms using carbon paper. However, this is very important in practical use, since this is still the most widely used method of paying with a credit card.

It is very easy to manipulate embossed characters, since the plastic can easily be flattened by moderately heating the embossed characters (using an iron, for example). In order to counter

this, one of the embossed characters is often placed on top of the hologram, which will be destroyed if it is heated.

Thermochrome displays

There are certain applications in which it is desirable to occasionally change the text and image(s) printed on the card. A good example is a student identification card in the form of a smart card that must be renewed twice a year. Ideally, it should be possible to visually read the expiry date without having to use any special equipment. This means that it must be printed on the card, rather than just being stored in the chip. A similar example is an electronic purse smart card, which requires using a card reader to show the current balance of the 'money' stored in the card.

Smart cards with microcontroller-driven displays are currently technically possible, but they are still too expensive for large-scale use. A thermochrome display (TC display) is a simple alternative that has some drawbacks compared with 'real' display, but is inexpensive and already available. A TC display is a supplementary card component on which characters and images can be reversibly printed (printed and subsequently reprinted) using a special card reader.

The technical operating principle is relatively simple. The thermochrome strip consists of a thin film (10–15 μm) of a temperature-sensitive material laminated to the card. This material darkens when it is heated to 120 °C. A printing head with a resolution of 200 or 300 dpi, such as is used in thermal-transfer and dye-sublimation printers, is used to heat individual points on the thermochrome strip to form characters or an image. This darkened material can be changed back to a nearly transparent state by heating the entire strip, which amounts to erasing the strip.

The thermochrome process is currently the only economical manner to present time-varying information to the user on the surface of the card such that it can be read without using any special equipment. Its major disadvantages are that it is subject to fraud and that it requires a special card reader with a built-in thermochrome printer.

The MM technique

In 1979, the German banking industry decided to include a machine-readable security feature in all German Eurocheque (EC) cards. After various potential methods were tested, the MM technique (developed by the firm GAO) was selected as the security process for these cards. This security feature is still used in all German Eurocheque cards, even though they are now equipped with microcontroller chips. The objective of this security feature was, and still is, to prevent unauthorized copying or modification of the magnetic-stripe data.

The MM technique is a typical example of a secret and very effective security feature. It has been used for two decades in millions of cards. Its basic structure is summarized in an article by Siegfried Otto [Otto 82].

The name 'MM technique' comes from the German term *moduliertes Merkmal* (modulated feature), which can be understood to refer to a machine-readable substance that is incorporated in the interior of the card body [Mayer 96]. A card is verified by reading its MM code using a special sensor and passing the code to a security module called the 'MM box'. The MM box

were the only way to verify the genuineness of the cards. The embedded microcontroller in the new type of card, and the cryptographic procedures that it makes possible, have diminished the importance of these features. They are nevertheless still very important whenever the genuineness of a card must be verified by a person instead of a machine, since a person cannot access the chip without special equipment.

Here we can only describe the most essential and best-known security features used with cards in a highly condensed form. There are many other types of features, such as invisible markings that can only be seen with IR or UV illumination, magnetic codes and special printing processes using rainbow-colored inks. These features are technically very interesting, but unfortunately there is not enough room to describe all of them.

In the future, security features will be found not only on the cards but also in the chips. It is conceivable that 'security' chips could be used in the same way that bank-note paper is now used. Genuine bank notes cannot be printed without using real bank-note paper, which has specific features to show that it is genuine. In order to incorporate similar security features into chips, special chips with specifically modified hardware are necessary. A terminal can then measure the modification, which constitutes the 'feature' of the chip, and judge the genuineness of the chips from the result.

As an example of a hardware feature, suppose that computation of a fast cryptographic algorithm is implemented in supplementary hardware in a certain chip. The time required to compute a particular value could be made so short, due to the hardware implementation of the algorithm, that it would not be possible to perform the same computation using a software emulation in a different chip in an equally short time. A terminal could thus distinguish this chip from other chips by making a simple timing measurement.

There are now chips available with hardware features similar or identical to what has just been described. Naturally, they are not freely available, just as bank-note paper is not freely available. Of course, such hardware features are only suitable for very large-scale applications, due to the high cost of developing chip-specific hardware. The consequence of this, which is that such chips are almost invariably available from only one manufacturer with no possibility of an alternate source, is difficult for many card producers to accept. However, hardware-based security is an important component of the security architecture of a smart card system, and it is unfortunately not available for free.

3.2 THE CARD BODY

The materials, construction and production of the body of the card are effectively determined by the card's functional components, as well as by the stresses to which it is subjected during use. Typical functional components include:

- magnetic stripe
- signature panel
- embossing
- imprinting of personal data via laser beam (text, photo, fingerprint)
- hologram

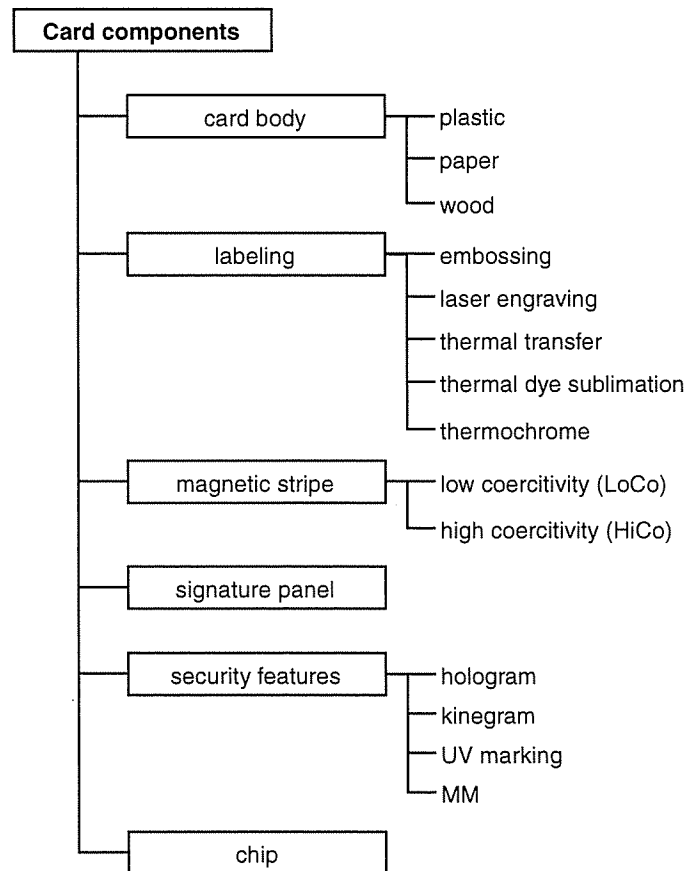


Figure 3.12 Classification scheme for card components

- security printing
- invisible authentication features (e.g. fluorescence)
- chip with contacts or other coupling elements

Clearly, even a relatively small card, only 0.76 mm thick, must sometimes contain a large number of functional components. This places extreme demands on the quality of the materials used and the manufacturing process. The minimum requirements relating to card robustness are specified in ISO standards 7810, 7813 and 7816 Part 1. The requirements essentially relate to the following areas:

- ultraviolet radiation
- X-ray radiation
- surface profile of the card
- mechanical robustness of the card and contacts

- electromagnetic susceptibility
- electrostatic discharges
- temperature resistance

The ISO/IEC 10373 standard specifies test methods for many of these requirements, to enable users and card manufacturers to objectively test card quality. The bending and twisting tests are particularly important for smart cards, since the chip, which is as fragile and brittle as glass, is a delicate foreign object in the elastic card. Special structural features are required to protect it against the mechanical stresses produced by bending and twisting the card. Chapter 9 contains a detailed list of tests and the methods used to perform them.

3.2.1 Card materials

The first material employed for ID cards, which is still widely used, is polyvinyl chloride (PVC), an amorphous thermoplastic material. It is the least expensive of all the available materials, easy to process and suitable for a wide range of applications. It is used throughout the world for credit cards. Its drawbacks are a limited lifetime, due to physical deterioration, and limited resistance to heat and cold. PVC is used in sheet form to manufacture cards, since injection molding is not possible. The worldwide production of PVC was around 13 million metric tons in 1996, of which 35,000 metric tons (0.27 %) were used for cards. PVC is considered to be environmentally hazardous, since the feedstock, vinyl chloride, is a known carcinogen. In addition, if it is burned, hydrochloric acid and (under unfavorable conditions) possibly dioxins are released. In addition, heavy-metal compounds are often used as stabilizers. Nonetheless, PVC is still by far the most widely used material for cards. This is primarily due to its low price and good processing characteristics. However, it used less and less each year due to its undesirable environmental properties. Many card issuers have decided not to use PVC for reasons of environmental policy.

To avoid the drawbacks of PVC, acrylonitrile butadiene styrene (ABS) has been used for some time to make cards. It is also an amorphous thermoplastic that is distinguished by its stability and resistance to temperature extremes. Consequently, it is often used for cards for mobile telephones, which for obvious reasons may be subjected to relatively high temperatures. ABS can be processed both in sheet form and by injection molding. Its major drawbacks are limited ink acceptance and low weathering resistance. Although the feedstock for ABS production, benzene, is a carcinogen, ABS has no other known environmental drawbacks.

For applications in which extreme stability and durability are required, polycarbonate (PC) is used. It is typically used for identity cards, and it is incidentally the base material for compact discs and DVDs. Due to its high thermal stability, relatively high temperatures are needed to apply holograms or magnetic stripes using the hot-stamp process. This can easily cause problems, due to the limited thermal stability of the materials being applied. The main drawbacks of polycarbonate are its low degree of resistance to scratching and very high cost compared with other card materials. A further drawback is that phosgene and chlorine are needed for the production of polycarbonate, and both of these materials are environmentally

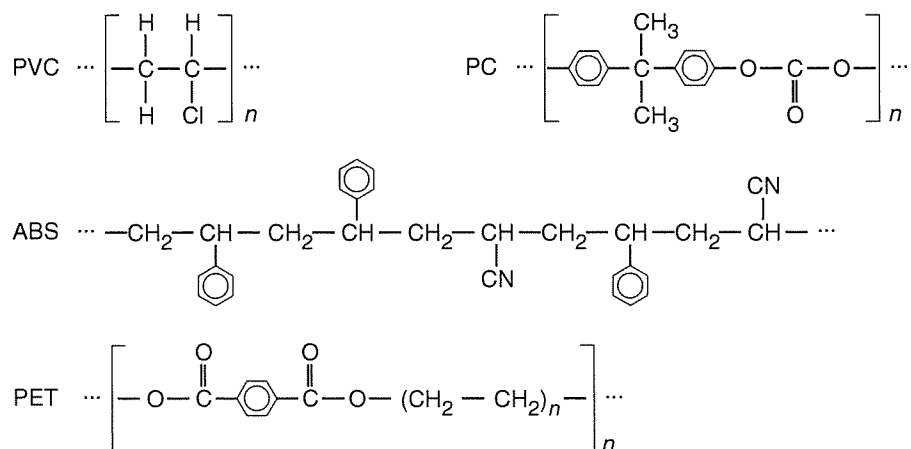


Figure 3.13 Structural formulae of the most important materials used for card bodies

problematical. Polycarbonate cards can be easily recognized by the characteristic ‘tinny’ sound they produce when dropped on a hard surface.

An environmentally friendly material that is mainly used as a PVC substitute is polyethylene terephthalate (PET), which has been used for a relatively long time to make packaging materials. It is commonly known as polyester. This thermoplastic material is used in smart cards in both its amorphous form (A-PET) and its crystalline form (PETP). Both types are suitable for processing in sheet form, as well as by injection molding. However, PETP is difficult to laminate, which makes additional processing steps necessary in the manufacturing process.

Numerous attempts have been made to find new or better materials for card bodies besides the usual materials (PVC, ABS, PC and PET). One example is cellulose acetate, which although having good environmental properties, has up to now proven to be poorly suited to the mass production of cards. Truly different materials, such as paper, have been frequently discussed, but as yet they have never been used in any significant quantity. The requirements imposed on cards, in terms of cost, durability and quality, are after all very high, and they can presently only be met by plastics.

In 1996 and 1997, Danmønt¹ conducted a field trial using around 600 cards that, while they did not represent a real alternative to plastic card bodies, were at least an interesting (or amusing) idea. The cards were laminated from eight layers of birch wood, each 0.1 mm thick. These cards did not meet the requirements of the various tests specified in ISO 10373, such as those for bending and twisting, and they were naturally not suitable for embossing. However, around 90 % of their users expressed a positive reaction and said that they experienced no problems with their cards. Unfortunately, a birchwood card is not especially innovative from an environmental perspective, since the layers must be laminated using a plastic adhesive and the usual printing processes are required.

¹ See [a la Card 97]

Table 3.1 Summary of the characteristics of the standard materials for card bodies.² The relative cost is based on the cost of PVC

Characteristic	PVC	ABS	PC	PET
Primary use	credit cards	cellular-telephone cards	identification cards	health-insurance cards
Principal feature	inexpensive	thermally stable	durable	environmentally friendly
Temperature range	65–95 °C	75–100 °C	<160 °C	<80 °C
Cold tolerance	moderate	high	moderate	moderate
Mechanical stability	good	good	good	very good
Embossing	good	poor	good	good
Printing	good	moderate	moderate	moderate
Hot stamping (e.g. holograms)	good	good	difficult	good
Laser engraving	yes	poor	good	good
Typical lifetime	≈2years	≈3years	≈5years	≈3years
Share of worldwide card production (1998)	85 %	8 %	5 %	2 %
Relative cost	1	2	7	2.5
Environmental aspects	stabilizers contain heavy metals burning may release dioxins	base material benzene is a carcinogen burning may release prussic acid	phosgene and chlorine needed for production burning does not release dangerous materials	the most environmentally friendly material burning does not release dangerous materials
Special features	negative public image		low resistance to scratching	

3.2.2 Chip modules

The most important component of a smart card is naturally the chip. Of course, this very fragile component cannot be simply laminated to the surface of the card like a magnetic stripe. Instead, it needs a sort of enclosure to protect it from the rough everyday life of the card. This enclosure is called the chip module. In addition to protection from ambient conditions,

² Based in part on [Houdeau 97, Grün 96]

chips for contact-type smart cards need six or eight contacts, which provide power to the chip and allow data communications with the terminal. A portion of the module's surface serves to provide these electrical contacts to the outside world. Naturally, the chip module should be as inexpensive as possible.

A wide variety of module designs have been devised in the course of the development of smart cards in order to meet these two technical requirements – protection of the fragile semiconductor chip and provision of contact surfaces. The most important of these are shown in Figures 3.14 and 3.15.

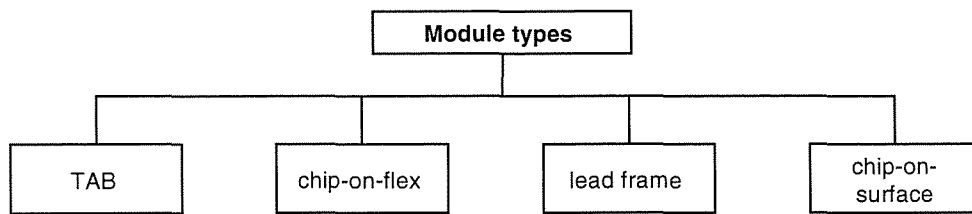


Figure 3.14 Classification of the various types of chip modules

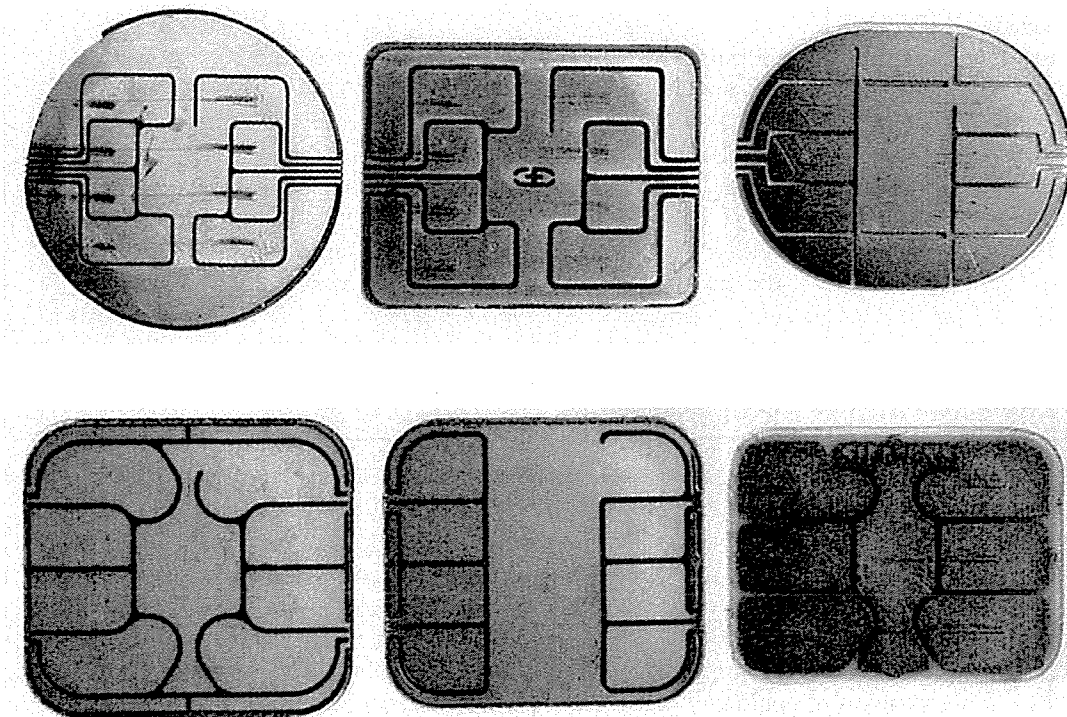


Figure 3.15 These examples illustrate the evolution of the chip-on-flex process, starting with one of the first eight-contact chip-on-flex modules at the upper left and proceeding to contemporary modules with six or eight contacts

3.2.2.1 *Electrical connections between the chip and the module*

Electrical connections are required between the chip inside the module and the contacts on the outside of the module. Presently, two processes are primarily used for this. In the wire-bonding process, an automatic bonding machine attaches gold wires with a diameter of only a few micrometers between the chip and the rear surfaces of the contacts. The wires are electrically attached to the chip and the module using ultrasonic welding. With this process, the contact arrangement on the top surface of the chip is always opposite that of module. This has been a standard process in the semiconductor industry for some time, and it can be readily used for mass-producing chip modules. However, each chip must be electrically connected to the module by five wires, which naturally costs time and money.

The die-bonding process was developed to further reduce the cost of fitting chips into modules. In this process, the electrical connections between the chip and module are not made with wires. Instead, the connections are made by mechanically attaching the chip to the rear surface of the module.

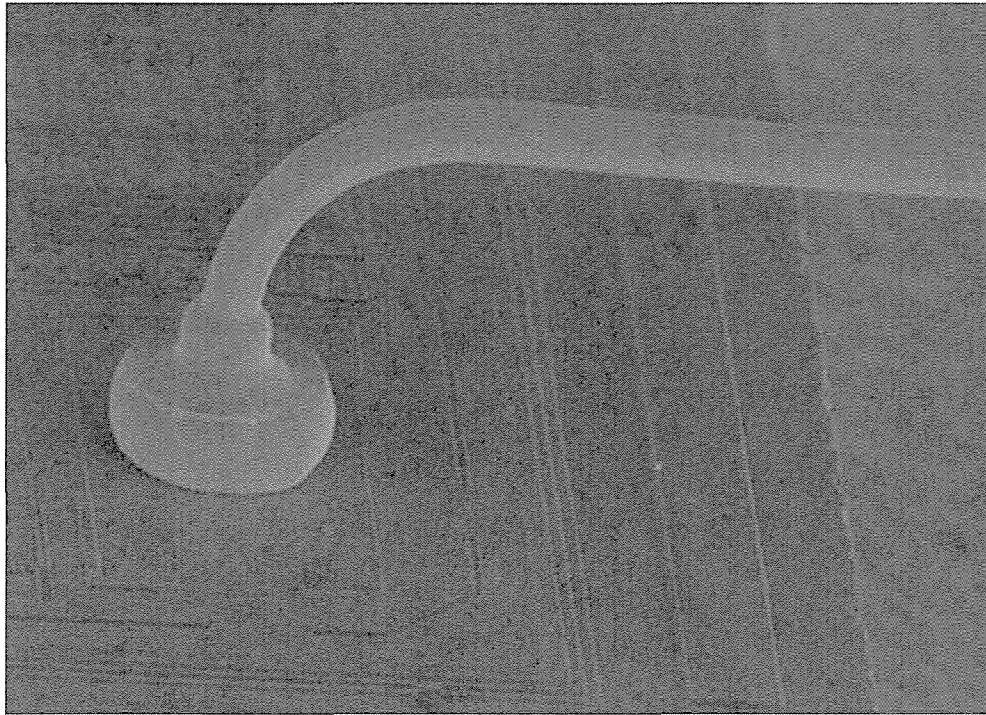


Figure 3.16 Photograph of the contact zone between a bonding wire and a bonding pad of a smart card microcontroller, magnified 1000 times (*Source: Giesecke & Devrient*)

3.2.2.2 *TAB modules*

Tape-automated bonding (TAB) was a standard process for large-volume chip packaging at the beginning of the 1990s, but it is presently not commonly used, since it has become technically obsolescent and too expensive. It is described here primarily for the sake of completeness.

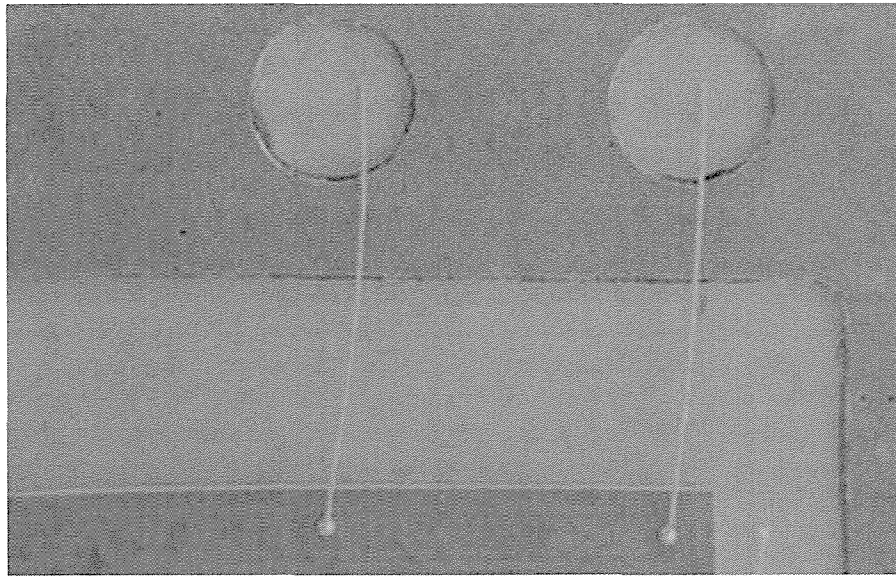


Figure 3.17 View of the electrical connections between a smart card microcontroller (bottom) and the chip module (top), magnified 400 times (*Source: Giesecke & Devrient*)

A chip module produced using the TAB process is shown in Figure 3.18. The special feature of this process is that metallic bumps are first electrically attached to the pads of the chip, and the leads of the carrier film are then soldered to these bumps. The solder connections are so sturdy that no additional support is required for the chip, which hangs from its leads. The active surface of the chip is protected against ambient conditions by an encapsulation material. The advantages of the TAB process are the mechanical strength of the connections to the chip and the low profile of the module. However, these advantages come at the price of higher costs compared with other module preparation processes.

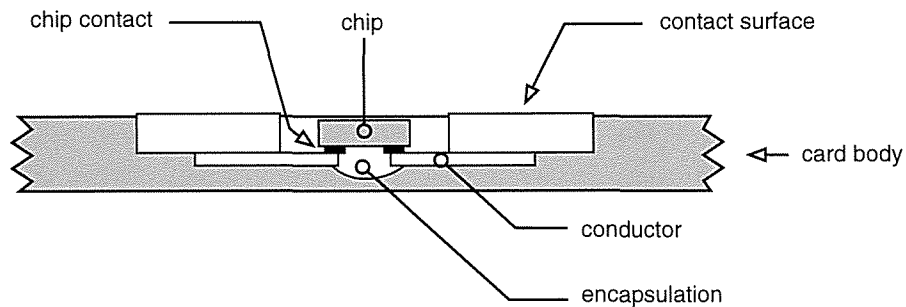


Figure 3.18 Cross-section of a chip module using the TAB process

Fitting a TAB module into a smart card is not easy, since the module must be taken into account in preparing the lamination foils for the card. Before the layers are laminated, suitable openings are punched in them, and the chip module is then inserted. The chip module is subsequently welded to the body of the card during the lamination process. This process provides a highly reliable bond between the chip module and the card body. It is nearly impossible to remove the chip from the card without destroying the card.

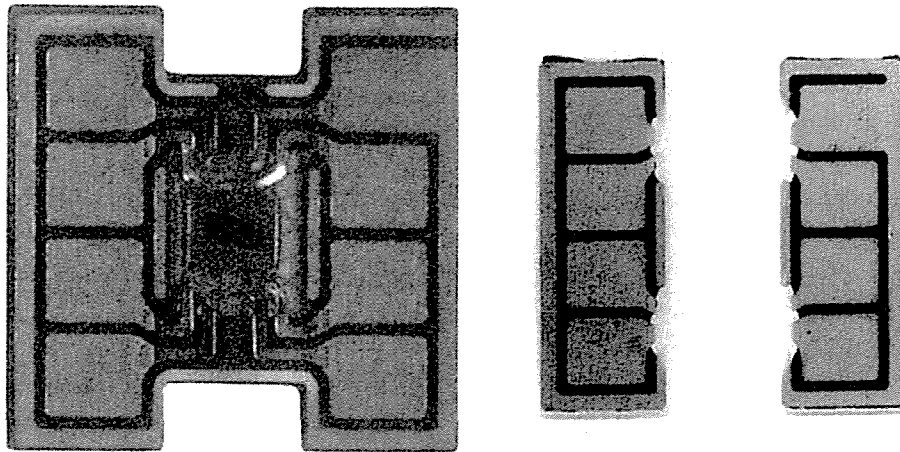


Figure 3.19 A TAB module ready for embedding in a smart card (left), and a TAB module fitted in a smart card (right)

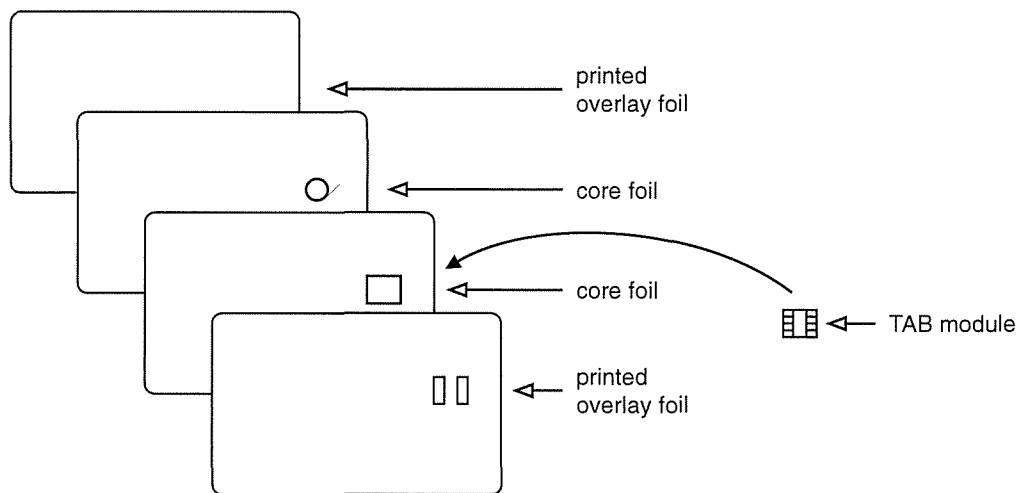


Figure 3.20 Inserting a TAB module during the lamination process

3.2.2.3 Chip-on-flex modules

Currently, the chip-on-flex module with wire-bonded contacts is the most widely used type of module. The construction of such a module is shown in cross-section in Figure 3.21. With this process, an opening into which the chip module can be glued is milled into the finished card body.

The carrier material is a flexible circuit board made of fiberglass-reinforced epoxy resin with a thickness of 120 μm . The contacts are formed from a layer of copper laminated onto the carrier, with a thickness of 35 or 75 μm . The contact surfaces are electroplated with gold in a later process step to protect them against processes that could adversely affect their electrical conductivity, such as oxidation. Holes are punched into the carrier to receive the chips and wire bonds. The chips, which are around 200 μm thick, are taken from the sawn wafer by a

pick-and-place robot and fitted into the openings in the circuit board. Next, the chip contacts are connected to the rear surfaces of the contacts using bonding wires a few micrometers in diameter. Finally, the chip and the bonding wires are encapsulated in a blob of synthetic resin to protect them against ambient conditions. The total thickness of the finished module is typically around 600 μm .

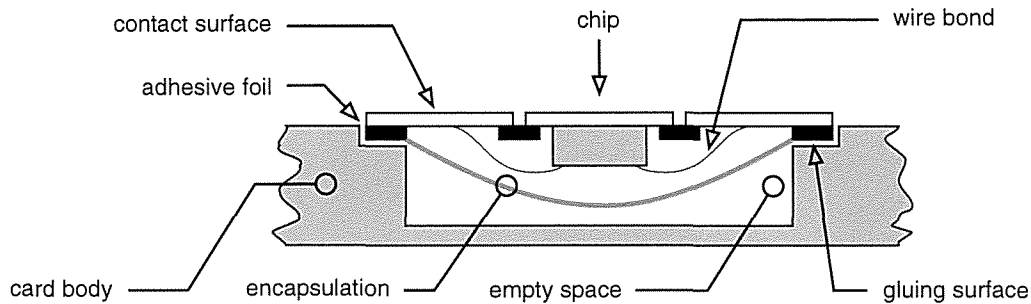


Figure 3.21 Cross-section of a chip-on-flex chip module

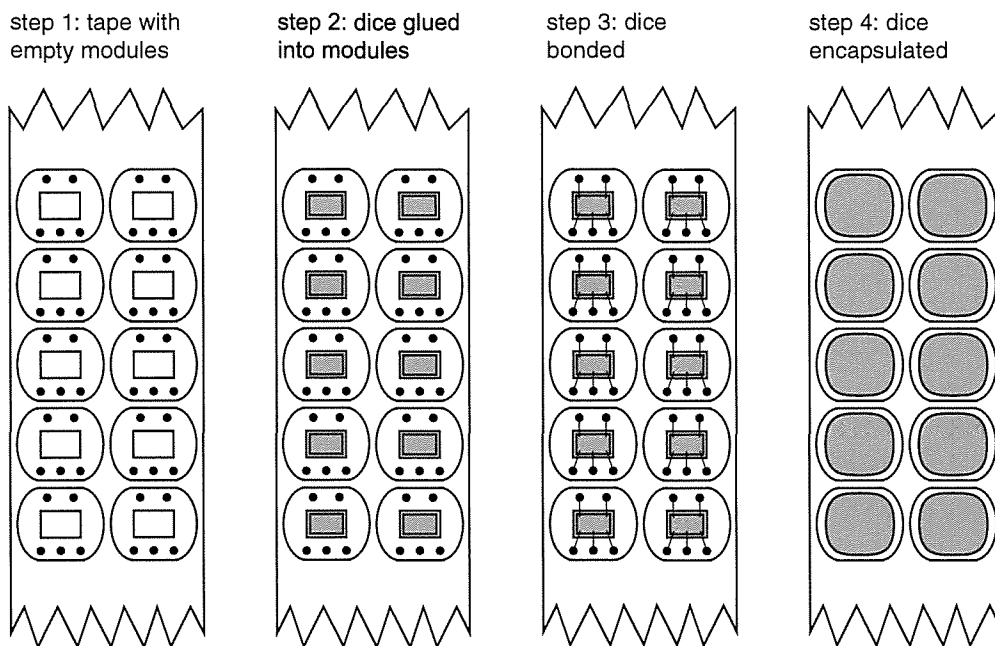


Figure 3.22 The four main process steps in the production of chip-on-flex modules

The advantage of this process is that it is largely based on a standard process used in the semiconductor industry for fitting chips in standard packages. It does not require as much specialized experience as the TAB process, so it is less expensive. This process also lends itself well to producing very complex card bodies with many active components. This is because defective card bodies can be separated from the rest before the expensive chip modules have been fitted. The disadvantage of this process is that the thickness and the surface dimensions of the chip module are significantly greater than those of a TAB module, since not only the chip

but also the bonding wires must be covered by the protective encapsulation. This is particularly disadvantageous, in that the standard smart card thickness of 0.76 mm does not leave a lot of room for overly thick modules.

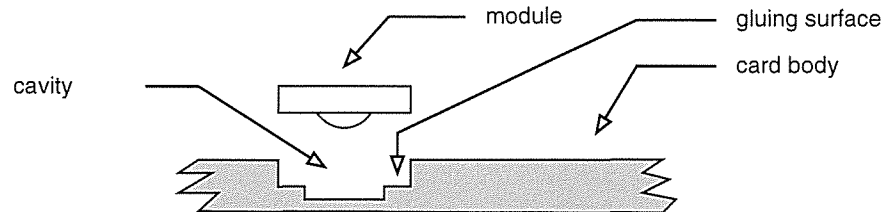


Figure 3.23 Inserting the chip module in a milled opening in the card body

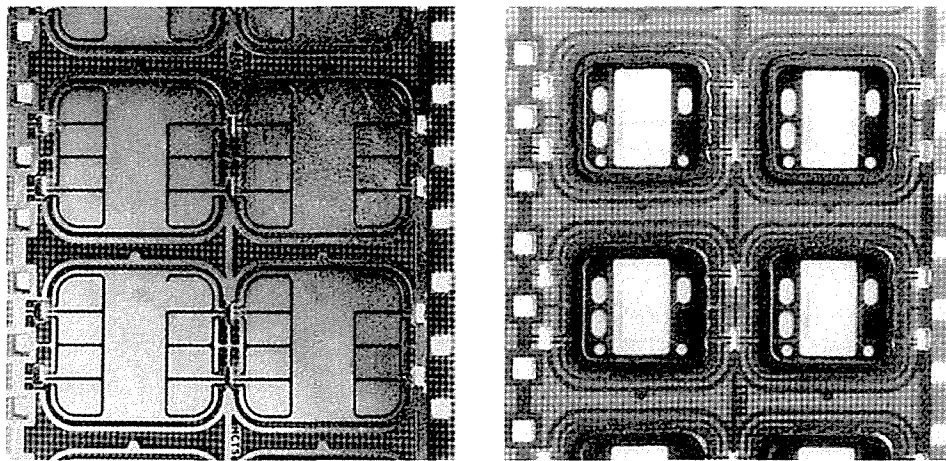


Figure 3.24 Front and rear views of chip-on-flex modules on 35-mm tape. The five openings in the carrier circuit board, for the bonding wires that make the electrical connections to the chip, can be clearly seen in the rear view

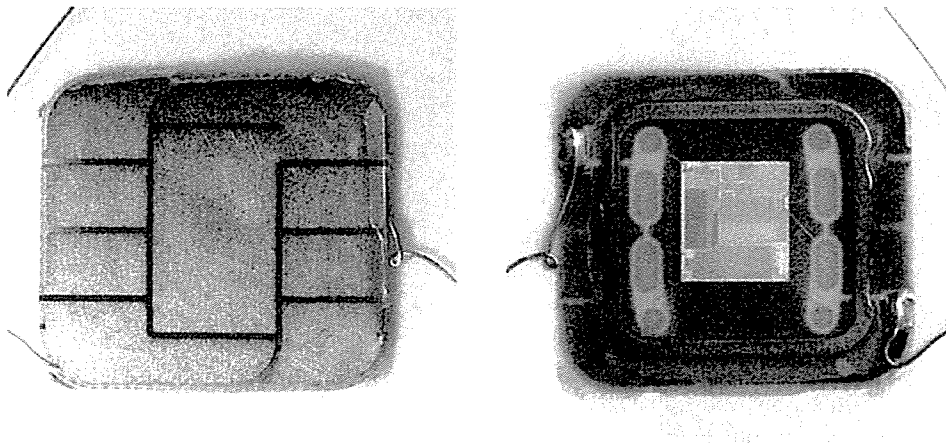


Figure 3.25 Front and rear views of a chip-on-flex module for a dual-interface card

3.2.2.4 Lead-frame modules

Technically, the TAB and chip-on-flex processes leave something to be desired, since they both provide little scope for reducing production costs. In the TAB process, producing the card body is very costly due to the characteristics of the module, while in the chip-on-flex process, the complexity of the module and the use of wire bonding lead to unfavorable production costs. These problems led to the development of a new type of module, the lead-frame module, which is mechanically just as robust as TAB and chip-on-flex modules but has lower production costs. The structure of a lead-frame module is relatively simple. The contacts, which are stamped from a gold-plated copper alloy, are held together by a plastic mold body. The chip is placed onto the lead frame by a pick-and-place robot and then connected to the backs of the contacts using wire bonding. Next, the chip is covered by a protective blob of opaque epoxy resin, usually black. The lead-frame process is currently one of the least expensive processes for making chip modules, without any accompanying reduction in the mechanical robustness of the modules.

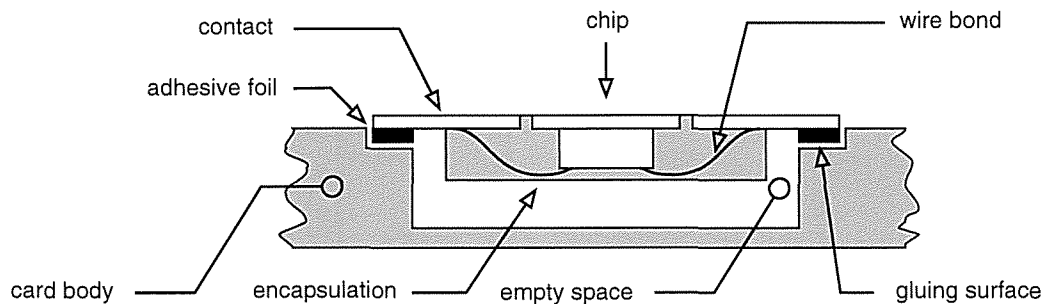


Figure 3.26 Cross-section through a lead-frame chip module

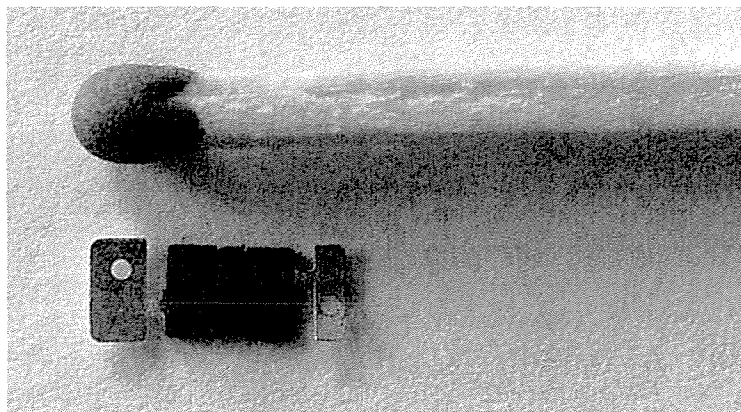


Figure 3.27 Stamped-out lead-frame module with the two coil connections for a contactless smart card, with a match for comparison

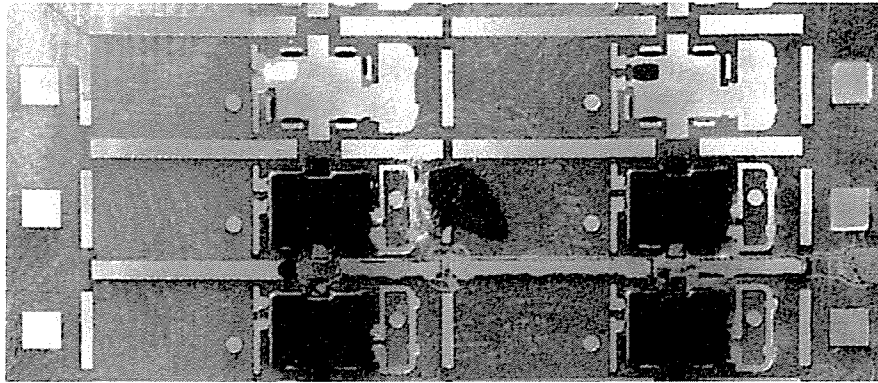


Figure 3.28 Lead-frame modules for contactless smart cards, arranged in pairs on a 35-mm tape. The two empty locations for modules that have been stamped out can be seen at the top

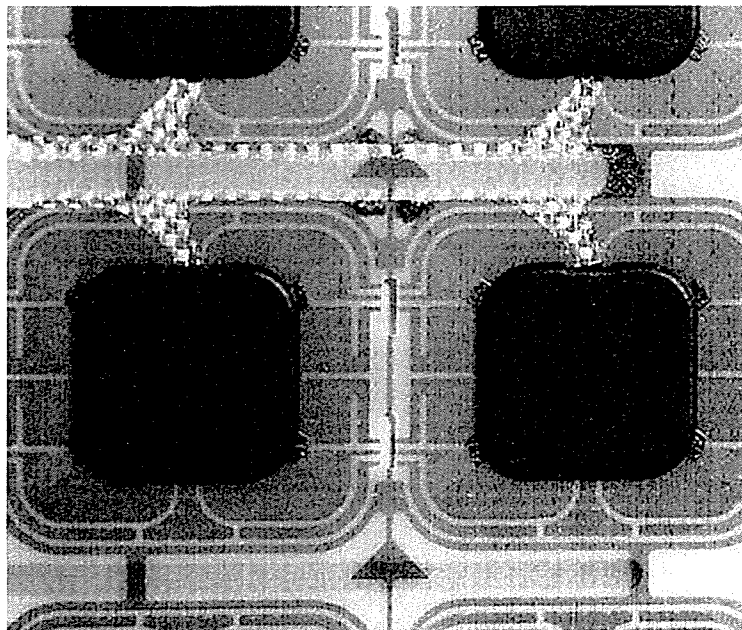


Figure 3.29 Lead-frame modules for smart cards with contacts, arranged in pairs on a 35-mm tape

3.2.2.5 The chip-on-surface process

For chips with relatively small surface areas, a process available since the mid-1990s offers a technically very interesting alternative to the usual process of fitting chips into modules. With the MOSAIC (Microchip on Surface and in Card) process, developed by Soliac [Sligos], no module is needed for the chip, since it is located directly in the card body.

The MOSAIC process is suitable for chips whose surface area is around 1 mm². This presently limits its application to pure memory chips, since microcontrollers are still too large for this process. The process works as follows: first, a laser is used to remove material from the location where the chip is to be placed, and then the chip is glued into this recess. In

the next step, a conductive silver paste is silk-screened onto the surface of the chip and the card body, thus forming contact surfaces and connecting them to the chip at the same time. In the final step, the chip and the leads to the contacts are covered with a non-conductive lacquer. This provides electrical insulation and protects them against external ambient conditions.

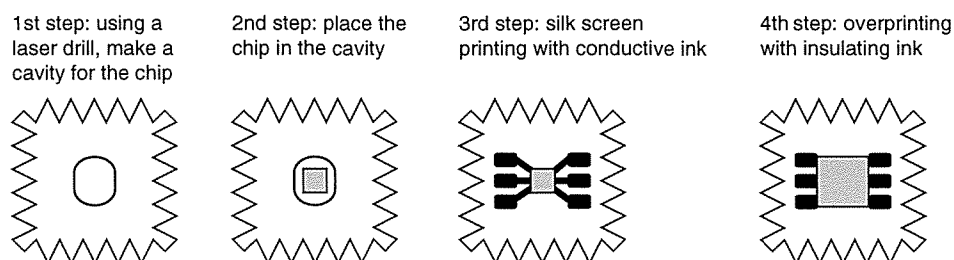


Figure 3.30 The four stages in the production of a smart card using the chip-on-surface process

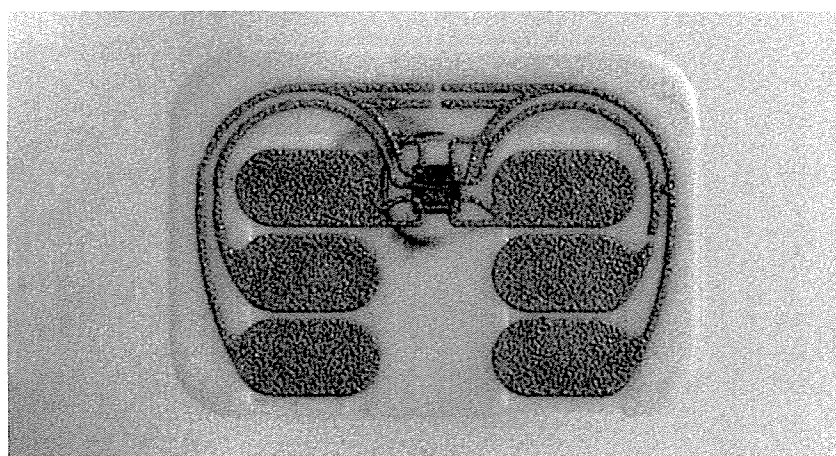


Figure 3.31 A memory chip with an edge length of 0.5 mm (0.25 mm^2 area) and ISO/IEC 7816-3 contact surfaces, fitted to a telephone card along with its contacts using the chip-on-surface process

As can clearly be seen from the figure, the chip-on-surface process is highly suitable for mass production of large numbers of cards, since it essentially consists of only a brief laser milling of the card body and two printing processes. However, this process requires an extremely precise printing process to ensure that the contacts for the chip are located correctly. Up to now, the card body has been primarily made of polycarbonate, which is especially suitable for the chip-on-surface process. The production capacity for finished cards lies in the region of 5000 pieces per hour per machine.

Another process is the flip-chip process in which the chip is placed with its face against the rear surface of the module and electrically bonded, after which the assembled module is filled with a casting resin. This type of low-cost module is usually referred to as FCOS (flip-chip on substrate).

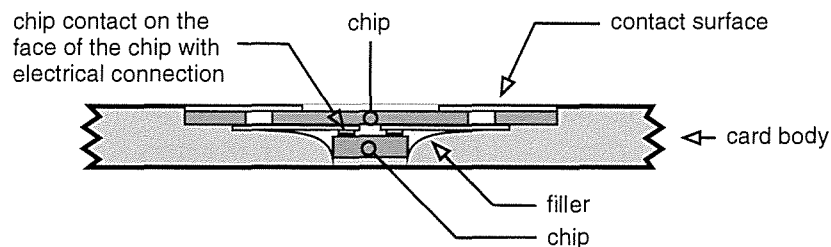


Figure 3.32 Cross-section of a chip module made using the flip-chip process

3.3 ELECTRICAL PROPERTIES

The electrical properties of smart cards depend solely on the embedded microcontroller, since it is the only component of the card with an electrical circuit. This situation will undoubtedly change in the future with the addition of other components to cards, such as displays, keypads and the like, but it will take some time before these new types of smart cards are widely used.

The application that from the very beginning has imposed many rigid requirements on the electrical properties of smart cards is mobile telecommunications using the GSM system. This system, which features an extremely large variety of technically different types of terminal devices made by an equally large variety of manufacturers, which must work with a variety of card types that is at least as large, has for a long time imposed extremely severe requirements. Due to the large number of smart cards used in the GSM system, the electrical characteristics specified for GSM cards have become general guidelines for all manufacturers of smart card microcontrollers. It can be assumed that nearly all new microcontrollers for smart cards will comply with the general electrical parameters of the relevant GSM specifications, since they otherwise would be practically unsellable in the telecommunications market.

In the early days of smart card technology, quite often the primary consideration was that the implanted microcontroller was functional, and less attention was paid to its general electrical properties, such as current consumption. At that time, the applications were almost exclusively closed, and they used a single type of card with a terminal specifically designed to match that type of card. The electrical properties of the smart card were relevant only in the sense that they had to be constant, since the terminal was designed to work with a particular type of microcontroller. However, the present situation is completely different. With current large-scale applications, in which various types of smart cards must work together with many different types of terminals, it is an unavoidable requirement that all of the cards that are used are either electrically identical or at least behave uniformly within clearly defined electrical regions.

The general international basis for the electrical properties of smart cards is the ISO/IEC 7816-3 standard and its associated amendment (Amd. 1). The amendment will be incorporated into the standard in the next major revision and thus disappear as a separate document. This standard specifies all of the fundamental electrical requirements for smart cards, such as the voltage ranges, maximum current consumption and the activation and deactivation sequences.

As usual with international standards, ISO/IEC 7816-1 provides a range of options that in many cases is too extensive for practical use. This allowed supplementary industry standards

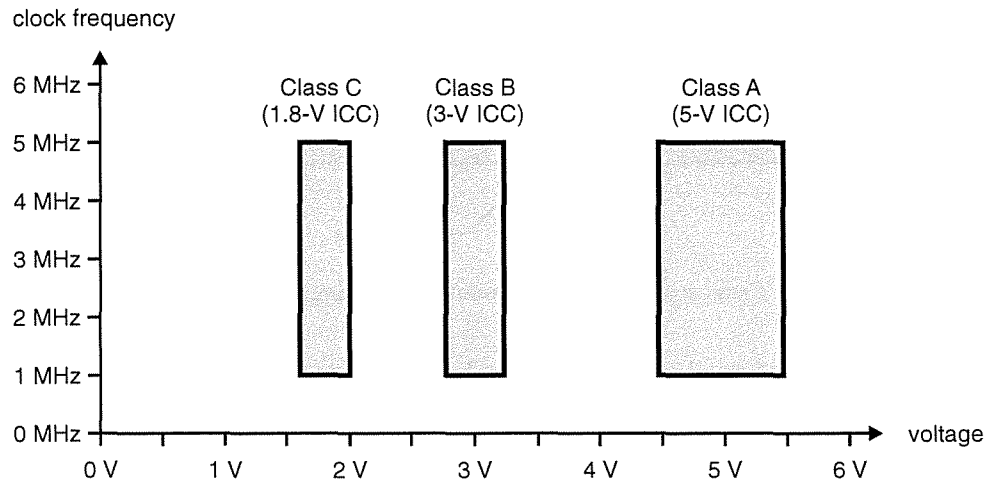


Figure 3.33 Comparison of the three possible classes of fixed voltage and clock frequency ranges as specified in ISO/IEC 7816-3 and ISO/IEC 7816-3 Amd. 1

to become established in the form of EMV 2000 for financial transactions and GSM 11.11, GSM 11.12, GSM 11.18 and TS 102.221 for telecommunications applications. These industry standards by no means compete with the ISO/IEC 7816-1 standard, but instead complement it with meaningful restrictions arising from practical smart card applications using millions of issued cards.

The distinction between smart cards used for financial transactions and smart cards used for telecommunications came about because certain requirements proved to have fundamentally different natures in these two application areas. For example, in the financial transactions area the current consumption and voltage range parameters are fully non-critical, since the terminals used for such applications are connected to the public power network. The situation in the telecommunications area is completely different, since every milliwatt counts when the objective is to achieve the longest possible operating time for a battery-powered mobile telephone. Consequently, the requirements for the least possible current consumption and low supply voltages are highly important in this application area.

Table 3.2 provides a summary of the most important electrical requirements of the essential international standards and industry standards. More detailed information is provided in the following sections.

3.3.1 Electrical connections

Smart cards have either six or eight contacts on the front side, which form the electrical interface between the terminal and the microcontroller in the card. All electrical signals are passed via these contacts. However, according to ISO/IEC 7816-2, two of the eight contacts (C4 and C8) are reserved for the auxiliary contacts AUX1 and AUX2, which can be used in the future for interfaces such as USB. Presently, some smart card modules have only six contacts, since this

Table 3.2 Summary of three electrical parameters (voltage, current and clock rate) for the most important international and industrial standards for smart cards. The tolerance ranges for the maximum current can be found in the relevant standards

Standard and class	Voltage	Clock rate	Maximum current
ISO/IEC 7816-3 and ISO/IEC 7816-3 Amd. 1			
Class A	5 V $\pm$ 10 % $\Rightarrow$ 4.5–5.5 V	1–5 MHz	60 mA at 5 MHz
Class B	3 V $\pm$ 10 % $\Rightarrow$ 2.7–3.3 V	1–5 MHz	50 mA at 4 MHz
Class C	1.8 V $\pm$ 10 % $\Rightarrow$ 1.62–1.98 V	1–5 MHz	30 mA at 4 MHz
Class A, B and C, with clock stopped			0.5 mA (clock stop)
EMV 2000	5 V $\pm$ 10 % $\Rightarrow$ 4.5–5.5 V	1–5 MHz	50 mA for all clock rates
GSM 11.11 5-V SIM	5 V $\pm$ 10 % $\Rightarrow$ 4.5–5.5 V	1–5 MHz	10 mA (operating) 200 μ A at 1 MHz (idle) 200 μ A (clock stop)
GSM 11.12 3-V SIM	3 V $\pm$ 10 % $\Rightarrow$ 2.7–3.3 V	1–5 MHz	6 mA at 3.3 V / 5 MHz (operating) 200 μ A at 1 MHz (idle) 100 μ A (clock stop)
GSM 11.18 1.8-V SIM	1.8 V $\pm$ 10 % $\Rightarrow$ 1.62–1.98 V	1–5 MHz	4 mA at 1.8 V / 5 MHz (operating) 200 μ A at 1 MHz (idle) 100 μ A (clock stop)
TS 102.221			
Class A, from reset to application selection	5 V $\pm$ 10 % $\Rightarrow$ 4.5–5.5 V	1–5 MHz	10 mA at 5 MHz (operating) 200 μ A at 1 MHz (idle)
Class A, during an application-specific session			60 mA at 5 MHz
Class B, from reset to application selection	3 V $\pm$ 10 % $\Rightarrow$ 2.7–3.3 V	1–5 MHz	7.5 mA at 5 MHz or 6 mA at 4 MHz (operating) 200 μ A at 1 MHz (idle)
Class B, during an application-specific session			50 mA at 5 MHz
Class C, from reset to application selection	1.8 V $\pm$ 10 % $\Rightarrow$ 1.62–1.98 V	1–5 MHz	5 mA at 5 MHz (operating) 4 mA at 4 MHz (operating) 200 μ A at 1 MHz (idle)
Class C, during an application-specific session			30 mA at 5 MHz

slightly reduces manufacturing costs. However, they have the same functionality as modules with eight contacts.

The contacts are numbered sequentially from top left to bottom right. Figure 3.34 shows the ISO designations and electrical assignments of the eight defined contacts.

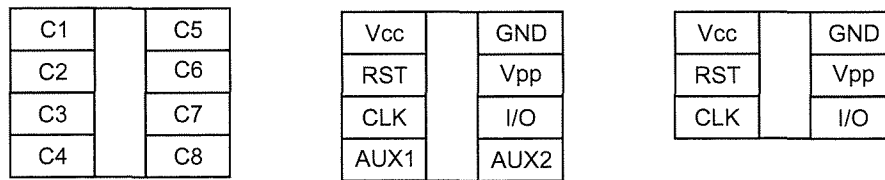


Figure 3.34 Electrical assignments and numbering of smart card contacts, per ISO 7816-2

Until the late 1980s, it was necessary to apply an external voltage to program and erase the EEPROM, since the microcontrollers then in use did not have charge pumps. Contact C6 was reserved for this purpose. However, since the early 1990s it has been standard practice to generate this voltage directly in the chip using a charge pump, so this contact is no longer used. Nevertheless, it cannot be employed for some other function, as this would conflict with the ISO standard. Thus, every smart card has a contact that has no real function, but which must still be present. Since the programming voltage contact lies between two others that are necessary for the operation of the card, it cannot simply be eliminated. This somewhat reduces the drawback of having a superfluous contact.

Table 3.3 Contact designations and functions according to ISO 7816-2

Contact	Designation	Function
C1	Vcc	Supply voltage
C2	RST	Reset input
C3	CLK	Clock input
C4	AUX1	Supplementary contact (Auxiliary 1)
C5	GND	Ground
C6	Vpp	Programming voltage (long since no longer used)
C7	I/O	Input/output for serial communications
C8	AUX2	Supplementary contact (Auxiliary 2)

3.3.2 Supply voltage

The supply voltage for smart cards was originally 5 volts, with a maximum tolerance of $\pm 10\%$. This voltage, which is the same as that used for conventional TTL circuits, was the standard value for all commercial smart cards and all applications.

As with other semiconductor components, increasingly smaller structure widths of semiconductor components and the need for reduced current consumption has made it necessary to markedly reduce the operating voltage range. This has been given extra impetus by the mobile telephone sector. The market-driven demand for reducing the weight of mobile telephones required changing from 6-V batteries to 3-V types, and since all other components for mobile telephones were available in 3-V technology, for a while the smart card was the only component in a mobile telephone that still needed 5 V. Consequently, an expensive voltage converter

was needed to provide electrical power to the smart card, resulting in an avoidable extra cost.

Consequently, in the international standards the voltage range for smart cards was first extended to 3–5 V with a tolerance of $\pm 10\%$. This yields an effective range of 2.7 V to 5.5 V. However, it can already be foreseen that this extension will be insufficient. Consequently, the revised version of ISO/IEC 7816-3 and ISO/IEC 7816-3 Amd. 1 will again be revised in the relatively near future to permit smart cards using a supply voltage of 1.8 V with a tolerance of $\pm 10\%$.

The extended voltage range does not pose a problem for the microprocessor or most types of memory, particularly since the core voltage for semiconductors built with 0.13- μm technology is usually only 1.8 V. However, EEPROMs are also integrated into smart card microcontrollers. These EEPROMs and their associated charge pumps form the greatest obstacle to low-voltage smart cards. Nevertheless, with a certain amount of technical ingenuity it is certainly possible to integrate EEPROMs and their charge pumps into microcontrollers that can work over a supply voltage range of 1.62 to 5.5 V.

The ISO/IEC 7816-3 standard and its amendment define three classes for characterizing the voltage ranges of smart cards. Class A covers the voltage range of $5\text{ V} \pm 10\%$, Class B covers the range of $3\text{ V} \pm 10\%$ and Class C covers the range of $1.8\text{ V} \pm 10\%$. All three classes can be used individually or in any desired combination. For instance, if a smart card meets the requirements for both Class A and Class B, it can be used with both 5-V and 3-V supply voltages. However, it must be borne in mind that the range between 3.3 V and 4.5 V lies outside the specified ranges, so the smart card need not necessarily be able to operate in this range. Nevertheless, smart cards can usually be used without any problems between the upper and lower limits of the specified voltage ranges.

The ISO/IEC 7816-3 standard imposes yet another equally important requirement, which is that under no circumstances may the microcontroller of a smart card be damaged if the card is powered from voltage not supported by the microcontroller. This is an essential requirement for ensuring the upward compatibility of new types of smart cards with older types of terminals. The objective is to eliminate the possibility that using a 3-V card in a 5-V terminal, for example, could destroy the IC in the card.

The three possible voltage ranges defined by ISO/IEC 7816-3 and ISO/IEC 7816-3 Amd. 1 have not yet been used in the area of smart cards for financial transactions. In this area, the original operating voltage of $5\text{ V} \pm 10\%$ still prevails, since stationary terminals can easily provide a voltage of 5 V without additional technical components.

The situation for smart cards used in the telecommunications area is completely different. In this area, smart cards that can be used only with 5-V supplies ('5-V-only cards') have fully disappeared. Since the end of the 1990s, 3 V has become the standard operating voltage for GSM devices. In the area of the new UMTS mobile telephone network, it is already clear that there will not be any mobile telephones that support 5 V, with the 3-V supply voltage being supported only for reasons of compatibility. In the medium term, the future standard operating voltage will be 1.8 V.

The ISO/IEC 7816-3 standard specifies a particular procedure for selecting the supply voltage, which essentially amounts to trying the voltages for each of the three classes in turn. As soon as the terminal can receive an ATR, it analyzes the ATR to see whether the smart card prefers a particular class. If so, the terminal initiates a new activation sequence using the desired class. If the ATR does not include any information about the voltage range, the smart

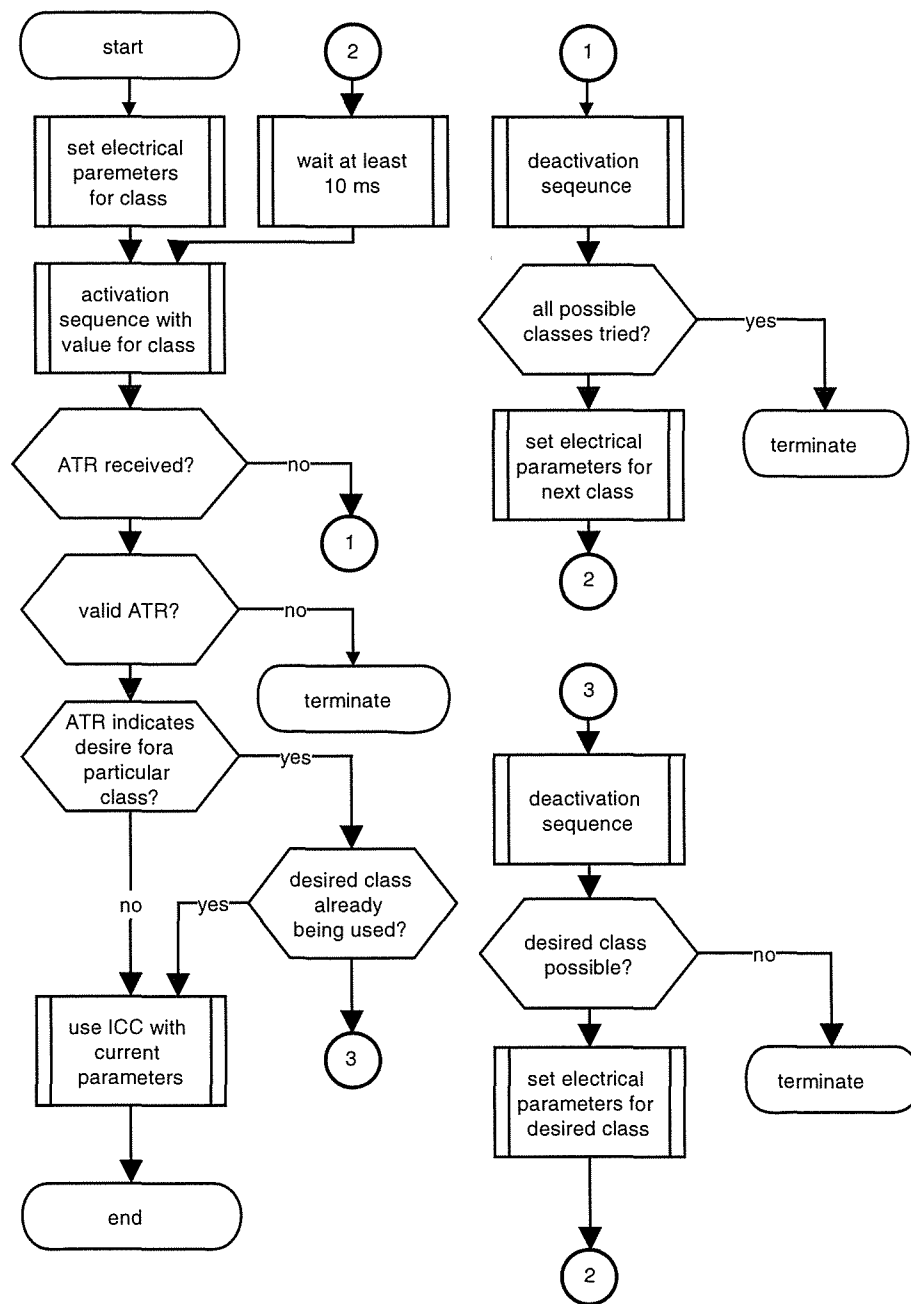


Figure 3.35 Flow chart showing the actions taken by a terminal when selecting the operating voltage based on the classes specified in ISO/IEC 7816-3 and ISO/IEC 7816-3 Amd. 1. With mobile end-user equipment, the process is usually started using the lowest supply voltage

card will be used with the voltage with which the first ATR could be received. The selection procedure is shown in the form of a flow chart in Figure 3.35.

It is already certain that an additional voltage class will be introduced in the future to allow a supply voltage of 1.2 V. Particularly in the light of the steadily decreasing structure widths

of semiconductor devices, in the medium term this could become a typical operating voltage for microcontrollers.

3.3.3 Supply current

The card's microcontroller obtains its supply voltage, and thus its supply current, via contact C1. According to the GSM 11.11 specification, this current may not exceed 10 mA. The current must lie below a certain value to allow the hardware in the terminal to be designed to supply a corresponding maximum current. The first version of the ISO/IEC 7816-3 standard in 1989 specified a maximum current of 200 mA with a 5-V supply voltage and 5-MHz clock, but even then that was too much. Since that time, the values have been significantly reduced and made dependent on the various supply voltage classes.

The most important factor is that the current consumption of a microcontroller is directly proportional to both the applied clock frequency and the supply voltage. It is also somewhat dependent on the temperature of the microcontroller. The current version of ISO/IEC 7816-3 specifies a maximum current of 60 mA for voltage class A (5 V) at a maximum clock frequency of 5 MHz and a maximum ambient temperature of 50 °C.

With regard to smart cards for financial transactions, in the EMV 2000 specification the ISO/IEC 7816-3 value for the maximum current is reduced from 60 mA to 50 mA, but there are no other significant supplementary restrictions. In the telecommunications sector, current consumption has been a critical factor since the very beginning. Consequently, in this sector there is a complicated set of rules specifying the maximum current as a function of the clock rate and operating state of the smart card. A detailed presentation of the maximum current for the various voltage classes is given in Table 3.2.

A technically interesting innovation has been introduced for smart cards that conform to the USIM specification. Here two different operating states are specified with regard to current consumption. The first state encompasses the time from the reset until selection of the application, while the second state comprises the subsequent application-specific session. The maximum allowable current consumption in the first state is significantly lower than in the second state. Furthermore, the mobile telephone can use an application-specific data object to determine the current demand of the currently selected application. This is because the current consumption of a smart card is considerably higher when its numeric coprocessor or internal frequency multiplier is enabled in order to achieve higher performance. With this mechanism, it would at least be theoretically possible to have a mobile telephone use only those 'smart applications' whose current consumption it can adequately support. Unfortunately, the corresponding USIM specification does not include any procedure to allow a mobile telephone and a smart card to negotiate the maximum available current, as with PPS. With the current version of the specification, the only option available to a mobile telephone if its smart card demands too much current is to deactivate the smart card.

Modern microcontrollers for smart cards have current consumptions on the order of 350 µA per megahertz of clock frequency. Using this value, we can write the following formula for the current consumption of a microcontroller as a function of the applied clock frequency or the clock frequency generated inside the chip:

$$I = \frac{f}{2.875} \cdot \frac{\text{mA}}{\text{MHz}}$$

This formula is useful for making initial estimates, but it must be remembered that the current consumption depends not only on the clock frequency, but also on the supply voltage, the temperature and of course the type of chip.

With a supply voltage of 5 V and an assumed current consumption of 60 mA, a smart card has a power consumption of 300 mW. This value is so low that there is no need to be concerned about overheating of the chip while it is operating, even though this amount of power is dissipated over an area of approximately 20 mm².

All smart card microcontrollers have one or more special power-saving modes. The operating principle of such modes is based on disabling all of the functional components of the chip that are not being used. In principle, only the interrupt logic of the I/O interface, the processor registers and the RAM need to remain energized in order to save the current operating state. In practice, the processor often remains energized as well, but the ROM and EEPROM are switched off. When the microcontroller is in this sleep mode, or idle state, its current consumption drops dramatically, since most parts of the chip are isolated from the supply voltage. In addition to this sleep mode, many smart card microcontrollers support another mode in which the applied clock can be switched off, called the 'clock stop mode'. The main purpose of this mode is to allow the hardware components in the terminal that generate the clock to be switched off, which makes this mode particularly attractive for battery-operated terminal devices. According to ISO/IEC 7816-3, the maximum allowable current in the sleep mode with the clock stopped is 500 µA for all three classes. Even this value is too high for the mobile telecommunications area. For instance, GSM 11.11 specifies an upper limit of 200 µA for 5-V smart cards at a clock frequency of 1 MHz.

current consumption

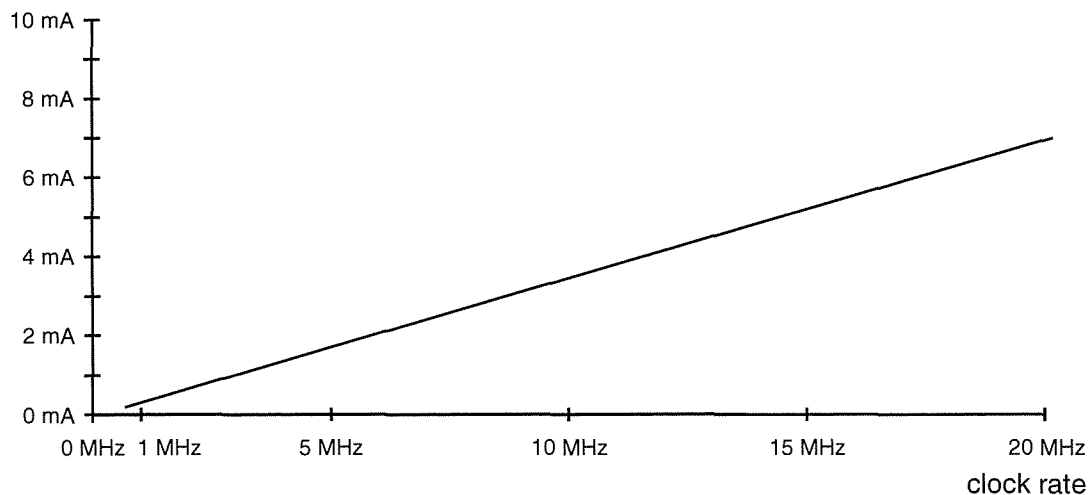


Figure 3.36 Microcontroller current consumption versus clock frequency in the normal operating mode (not the sleep mode). The current consumption in the sleep mode with the clock applied is also linearly dependent on the clock frequency and is approximately 50 µA at 5 MHz, depending on the microcontroller type

Another important detail regarding the supply current causes severe headaches for terminal manufacturers who choose to ignore it. All current microcontrollers employ CMOS

technology. Under certain conditions, large short-circuit current can occur briefly during transistor switching processes. These produce current spikes that are many times greater than the nominal operating current, with durations in the nanosecond range. These spikes can also occur when the EEPROM charge pump switches on. If the terminal cannot supply such large currents during these short intervals, the supply voltage will drop below the permitted value. This can produce a write error in the EEPROM or trigger the undervoltage detector in the chip.

For this reason, references to such spikes can now be found in practically every relevant standard and specification. For instance, ISO/IEC 7816-3 requires power sources for class-A (5-V) cards to be able to handle spikes with a maximum duration of 400 ns and a maximum amplitude of 100 mA. Assuming a triangular spike, this amounts to a charge of 20 nA-s that must be supplied. This requirement can be met in a simple manner by connecting a 100-nF ceramic capacitor between circuit ground and the supply voltage line very close to the contacts for the card.

3.3.4 External clock

Smart card processors usually do not have internal clock generators. An externally supplied clock is therefore necessary. This clock also provides the reference for data transmission rates. According to ISO/IEC 7816-3 and most other standards and specifications, the duty factor of the clock must be 50 %. The usual tolerance is a duty factor range of 40 to 60 %.

The clock signal applied to the contact is not necessarily the same as the internal clock provided to the processor. Some microcontrollers have a clock multiplier or divider that may optionally be inserted between the external and internal clocks. The clock divider frequently has a division factor of 2, so the internal clock rate is only half of the external clock rate. This is partly due to the characteristics of the chip hardware and partly because it allows oscillators already present in terminals to be used as the source of the clock signal for the chip.

Most smart card microcontrollers allow the clock signal to be switched off when the CPU is in the sleep mode. In this case, switching off the clock means holding the clock line at a defined level. Depending on the preference of the chip manufacturer, the 'off' level may be either high or low.

Since smart cards draw only a few microamperes from the clock line, switching off the clock may at first glance appear somewhat curious. Nevertheless, the amount of power saved within the terminal is substantial, so it can be worthwhile in certain applications.

3.3.5 Data transmission

If an error occurs during data transmission, it may happen that the terminal and the card attempt to send data at the same time. This results in a data collision on the connecting I/O line. Quite apart from the problems this causes at the application level, at the physical level it could produce currents in the I/O line that might be large enough to destroy the interface components. To prevent damage to the semiconductors in such an event, the I/O line in the terminal is tied to the +5-V level via a 20-k Ω pull-up resistor, as shown in Figure 3.37. In

combination with the agreed convention of never sending an active 5-V level, this avoids any problems that might occur if the two parties attempted to drive the data line to two different levels as the result of a communications error. Whenever the I/O line has to be set to a +5-V level during communications, the party in question simply switches its output to a high-impedance state (tri-state level), and the line is raised to the +5-V level by the pull-up resistor alone.

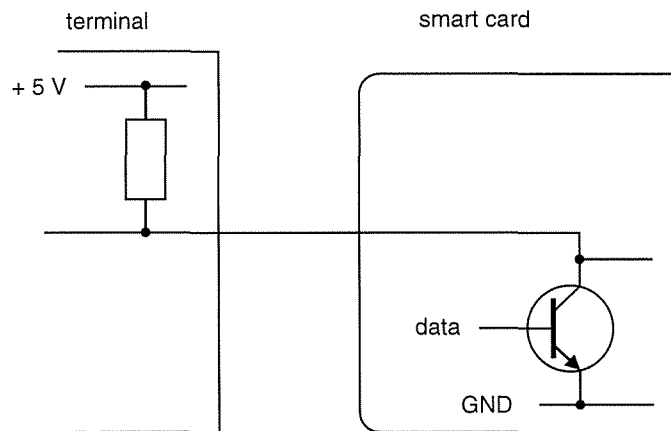


Figure 3.37 The circuit of the I/O channel between the terminal and the smart card

3.3.6 Activation and deactivation sequences

All smart card microcontrollers are protected against electrostatic charges and potentials on the contacts. In order to avoid undefined states, precisely specified activation and deactivation sequences are prescribed, and they must be strictly adhered to. This is also reflected in the relevant part of ISO/IEC 7816-3. These sequences define the electrical aspects of activating and deactivating the card and have nothing to do with the sequence of establishing mechanical contact with the card, which is anyhow not specified. Nevertheless, mechanical contact is first made with the ground contact of the card as an intelligent precaution in order to ensure well-defined electrical connection and disconnection.

As shown in Figure 3.38, the electrical ground connection must be made first, followed by the supply voltage connection. After this comes the clock connection. If an attempt were made to connect the clock before the supply voltage, for example, the microcontroller would try to draw its entire supply current via the clock line. This could irreversibly damage the chip, causing complete functional failure. A faulty deactivation sequence could also have similar effects on the microcontroller.

When the microcontroller is operating, it can be reset via the reset line. This requires a low level to be first applied to this line, with the actual reset being initiated by the subsequent rising edge. Such a reset during operation is called a warm reset, as in other computer systems. By contrast, a cold reset is one that occurs when all the supply lines are switched as specified in the ISO standard.

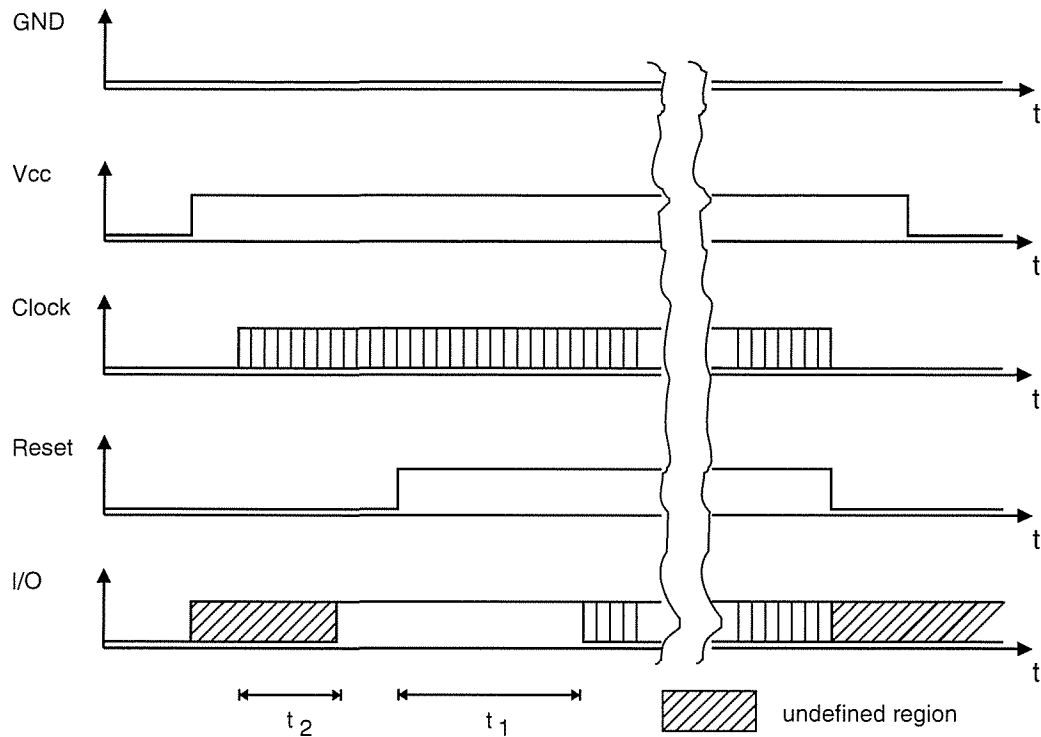


Figure 3.38 Smart card activation and deactivation sequences according to ISO/IEC 7816-3. The intervals t_1 and t_2 lie in the ranges $400/f \leq t_1 \leq 40,000/f$ and $t_2 \leq 200/f$

3.4 SMART CARD MICROCONTROLLERS

From an informatics perspective, the central component of a smart card is the microcontroller embedded under the contacts. It controls, initiates and monitors all of the card's activities. The microcontrollers that have been specially designed and developed for this purpose are complete computers in their own right. This means that they contain processors, memory and interfaces to the outside world.

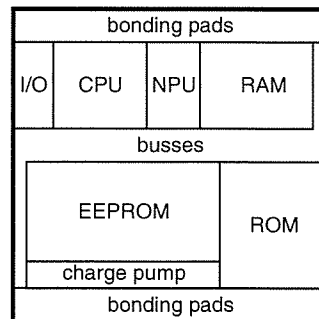


Figure 3.39 Possible arrangement of the essential functional components on the die of a simple smart card microcontroller

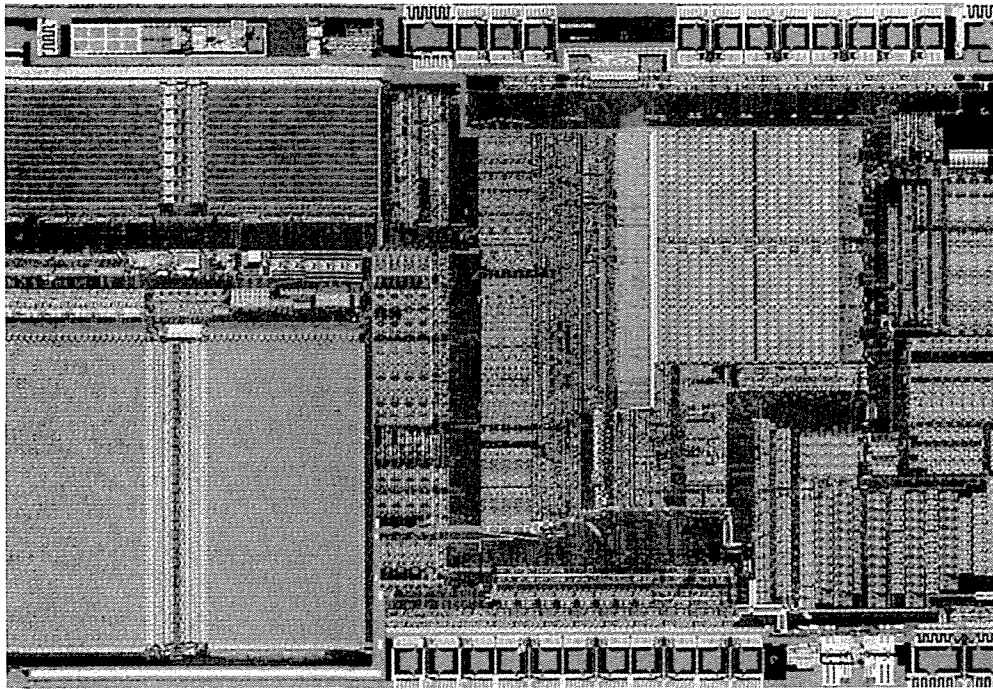


Figure 3.40 Photograph of a PC 83C852 smart card microcontroller with the following functional components (from top left to bottom right): ROM, EEPROM, processor with coprocessor and RAM. This chip has an area of 22.3 mm² and contains 183,000 transistors. Although this chip is no longer produced, it clearly shows the arrangement of the functional components on the die (*Source: Philips*)

The most important functional components of a typical smart card microcontroller are the processor, the address and data buses and the three types of memory (RAM, ROM and EEPROM). The chip also has an interface unit that provides serial communication with the outside world. This interface should not be imagined to be a complex functional unit that can independently transmit and receive data. In the simplest case, the serial interface is just a location that can be addressed by the CPU and is connected to the I/O contact.

In addition, some manufacturers provide special processors on the chip that act as a sort of mathematical coprocessor, although the functions provided by these components are limited to exponential and modulus operations on integers. Both of these operations are fundamental and necessary elements of public-key encryption procedures, such as the RSA algorithm.

The semiconductor technologies that are presently commonly used to produce smart card microcontrollers work with structure widths of around 0.25 μm , 0.18 μm and 0.13 μm , which definitely lie in the range of the smallest currently achievable structure widths.

Microcontrollers used in smart cards are not standard, widely available components. Instead, they have been specifically developed for this purpose, and they are not used in other applications. There are several important reasons for this, which are described below.

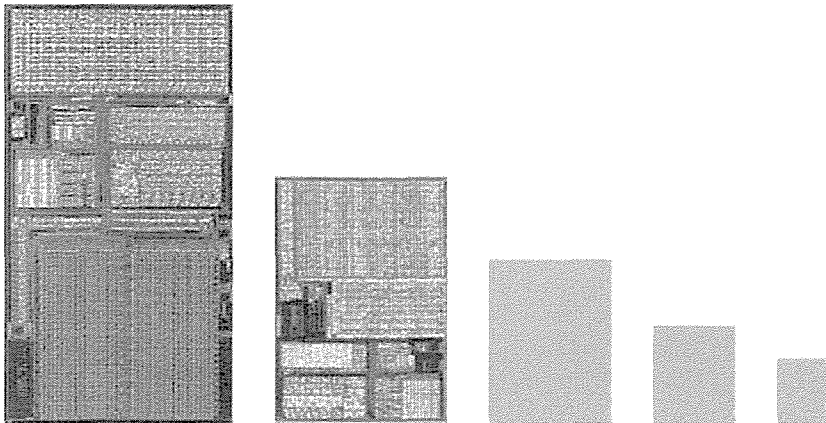


Figure 3.41 Relative sizes of functionally identical smart card microcontrollers before and after chip-area reduction ('shrink processing'). At the far left is an SLE 44C80 in 1- μm technology, with a surface area of 21.7 mm². To its right is an SLE 44C80S in 0.8- μm technology, with a surface area of 10 mm². The three gray rectangles show the relative sizes this smart card microcontroller would have if fabricated in 0.5- μm , 0.35- μm and 0.13- μm technologies (*Photos: Infineon Technologies*)

Manufacturing costs

The surface area of the microcontroller on the silicon wafer is one of the decisive factors with regard to manufacturing costs. A large chip area leads to more complicated packaging in the module, and thus increased costs. The chip area is thus kept as small as possible.

Furthermore, many commercially available standard devices include functions that are not needed in smart cards. Since these functions take up extra space on the wafer, they can be deleted from chips designed for smart cards. Although only a small reduction in the manufacturing cost per chip is achieved by such efforts to minimize the chip size, these small savings add up to a significant amount when a large number of chips are produced. This justifies the modifications to the chip design.

Functionality

Due to the need to integrate all the functional components of a computer into a single silicon chip, the available number of suitable semiconductor devices is extremely limited. Given the requirements of a minimum chip area, 5-V or 3-V supply voltage and a serial interface on the chip, all standard devices are effectively ruled out. In addition, the chip must contain a memory that can be written and erased but that does not require a permanent power supply for data retention (EEPROM or Flash EEPROM).

Security

Since smart cards are primarily used in security-related areas that require both passive and active security features in the chip, developing chips specially designed for this purpose is an unavoidable necessity.

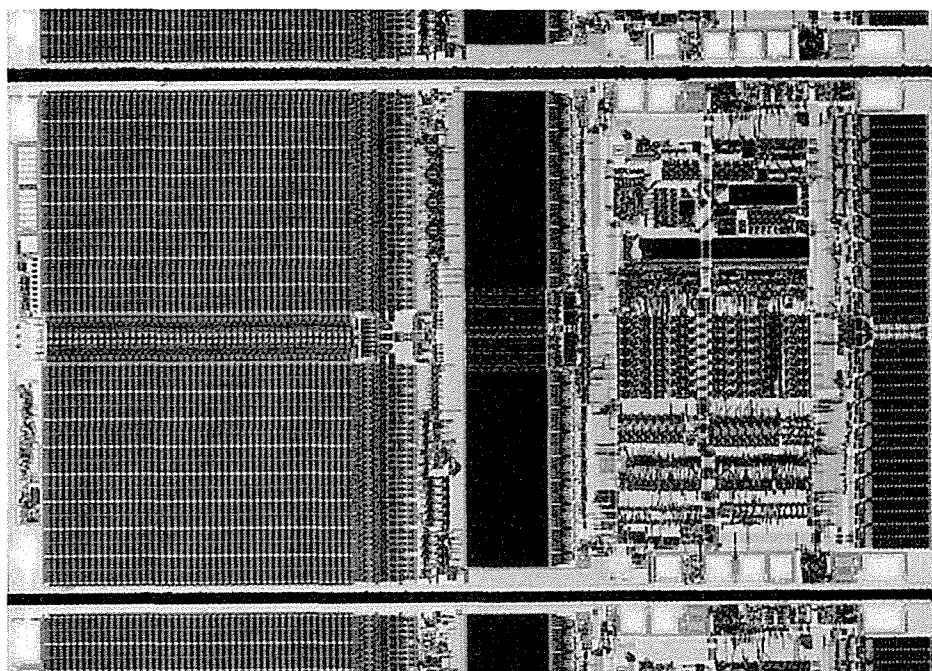


Figure 3.42 An ST16623 smart card microcontroller with the following functional components (from left to right): ROM, EEPROM, CPU and RAM. Although this chip is no longer manufactured, it clearly shows the arrangement of the individual functional components on the die
(Source: ST Microelectronics)

Chip area

The size of the microcontroller die strongly affects the fragility of the chip. A larger die is more likely to break when the card is bent or twisted. Consider a telephone card carried in a wallet, for instance: the bending stresses on the card and the embedded chip are enormous. Even the finest hairline crack in the chip is sufficient to render it useless. Therefore, most card manufacturers impose an upper limit of about 25 mm² on the chip area and demand that the layout be as nearly possible square, in order to minimize the risk of fracture.

Availability

The security policy of many card manufacturers is that the microcontrollers they use are not available on the open market. This makes it considerably more difficult to analyze the chip's hardware, since a potential attacker normally does not have access it.

However, this position has been seriously weakened by the general availability of programmable smart cards, such as Java Card types, and it is generally no longer defensible for standard applications.

Limiting the types of microcontrollers used to only a few specialized types (and consequently only a few manufacturers) has the disadvantage that the card manufacturer is highly dependent

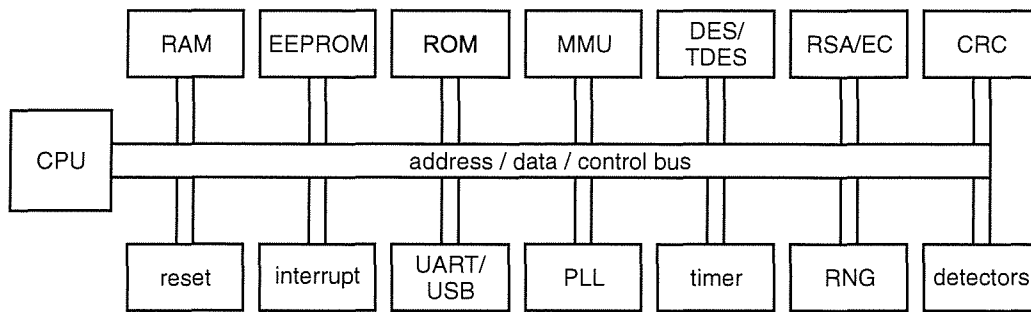


Figure 3.43 The essential functional logic units of a high-performance smart card microcontroller

on the chip suppliers. If a semiconductor manufacturer experiences production problems, it is not possible to quickly switch to a different device.

3.4.1 Processor types

The processors used in smart cards are not special designs, but instead proven devices that have been used in other areas for a long time. In this industry, it is not usual to develop new processors for special application areas, since this is generally too expensive. In addition, it would yield a completely unfamiliar processor, for which no suitable function libraries and development tools would be available from producers of operating systems.

Additionally, smart card processors must be extremely reliable. It is therefore better to rely on older type processors that have been proven in practice, rather than experimenting with the latest developments of semiconductor manufacturers. The aerospace industry, which is very interested in functional security, uses only components that are one or two generations behind the current state of the art, for the same reasons.

The transistor was invented at the Bell labs in 1947. Intel brought out the first microprocessor in 1971, with the type designation '4004'. It contained 2300 transistors and had a clock frequency of 108 kHz, 45 machine instructions, 604 bytes of address space and a 4-bit data bus, yielding a processing capacity of 0.06 MIPS. Since then, the development of integrated microprocessor components has made huge strides. This is clearly shown by recent products, such as the 32-bit Pentium IV processor with 42 million transistors on 217 mm² in 0.13-μm technology and a 2-GHz clock rate. However, the most technologically advanced processors are not used in smart cards, for the reasons that have just been stated. A total of 200,000 transistors is typical for an IC with a midrange processing capacity.

Smart card microcontrollers at the lower end of the performance scale usually have an addressable memory in the range of 6 KB to 30 KB. Under these conditions, using an 8-bit memory bus does not impose any significant restrictions. The processors used generally have a CISC (complex-instruction-set computer) architecture, which means that they require several clock cycles to execute machine instructions and usually have very large instruction sets. The address range of the 8-bit processors is most often 16 bits, which allows up to 65,536 bytes to be addressed. The processor instruction sets are based on either the Motorola 6805 or Intel 8051 architecture. The semiconductor manufacturer may add supplementary instructions to the standard instruction set. Such instructions usually involve additional options for 16-bit

memory addressing, which exists only in the most rudimentary form in the two instruction sets that form the basis for the instruction sets of smart card processors.

Processors in the 8-bit families are also available with extensions that allow them to address additional memory banks, in order to surmount the 64-kB limit. Access to such memory banks is controlled via special registers that map the memory banks into a specific memory region, where they can be accessed by the processor. However, this type of non-linear memory addressing has significant drawbacks. For instance, the relatively complex distribution of program code over several memory banks significantly complicates the software, thus increasing the likelihood of errors. Additional memory is also needed for bank switching functionality. Consequently, memory space expansion using memory banks is primarily a makeshift remedy that is used while awaiting the transition to processors with larger bit widths.

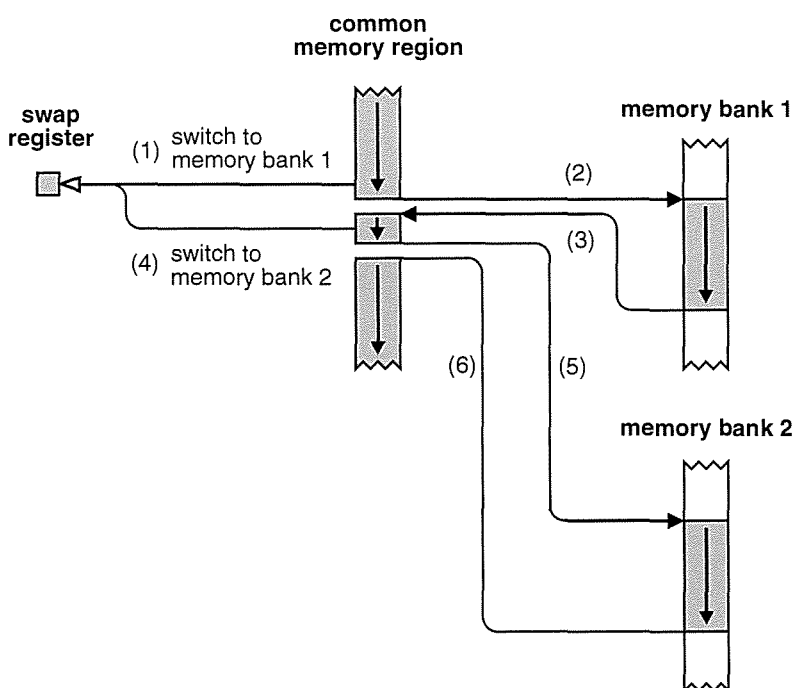


Figure 3.44 Basic arrangement of a memory divided into several banks and the associated program flow. This example shows two subroutines located in two different memory banks being called from a common memory region. The numbers in parentheses indicate the sequence of events in the process

The 16-bit processors include several derivatives of existing 8051 architectures, as well as company-specific designs, such as the Renesas H8, Philips XA and Samsung CALM devices. Incidentally, for a long time the H8 was the only 16-bit processor for smart card microcontrollers with a RISC-like architecture and a corresponding instruction set ('RISC' stands for 'reduced-instruction-set computer').

At the upper end of the performance scale for smart card microcontrollers, both 16-bit and 32-bit types are now available. The development trend is quite clearly heading in the direction of 32-bit processors. Such processors are urgently needed in this performance class, in order to handle large memories (exceeding the 64-kB boundary) and above all to satisfy the enormous processing appetites of modern interpreter-based smart card operating systems, such as Java

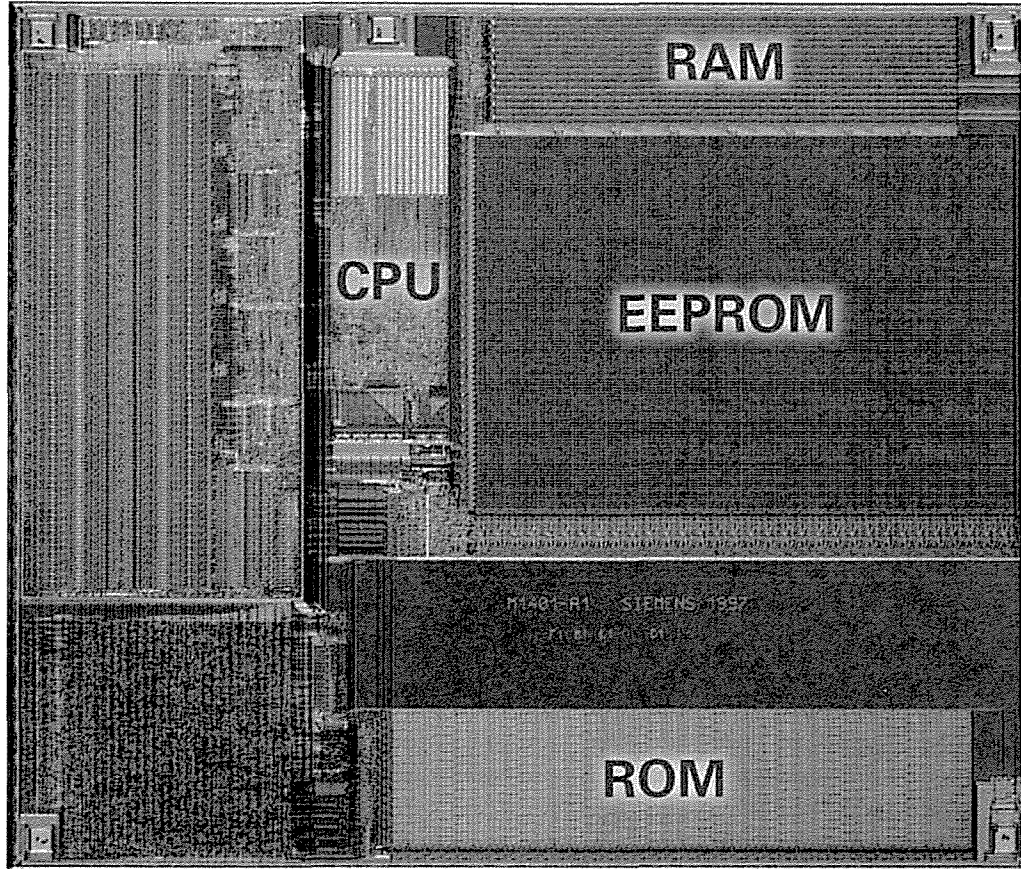


Figure 3.45 An SLE 66CX160S smart card microcontroller with an area of 21 mm², fabricated in 0.6-μm technology with 32 kB of ROM, 16 kB of EEPROM and 1280 bytes of RAM. The two unlabeled regions on the left-hand side of the chip are the numeric coprocessor and the peripheral components (timer, random-number generator and CRC coprocessor). The five bonding pads for the electrical connections to the module contacts can be clearly seen in the photo (*Photo: Infineon Technologies*)

Card. The key selection criteria for processors include code density, power dissipation and resistance to attacks.

Among the 32-bit processors, company-specific types are presently becoming established, but two processor cores used in typical microcontroller markets have also gained an entry into the smart card realm. They are the MIPS [MIPS] and ARM [ARM] processors.

The first step into the 32-bit league was taken in 1993 with the European CASCADE (‘Chip Architecture for Smart Card and portable intelligent Devices’) project. One objective of this project was to provide a high-performance processor for smart cards. The project selected the ARM 7M RISC processor, which is often used in portable equipment such as video cameras and PDAs.³ It has a 32-bit architecture and can run at up to 20 MHz with a 3-V supply voltage,

³ Based on [Peyet 97]

while drawing only 40 mA. In 0.8- μm technology, the ARM 7 core has an area of 5.9 mm² (3.12×1.9 mm), with the associated arithmetic processor for the usual public-key algorithms (RSA, DSA and EC) occupying an additional 2 mm². The processor has both supervisor and user modes, and thus supports partitioning of the operating-system code and application code. Since originally being selected, the ARM 7 has been optimized for smart card applications, and in the form of the SC 100 ('secure core') it has been built into smart card microcontrollers by various semiconductor manufacturers. The next evolutionary step is a dedicated smart card variant of the ARM 9, with the type designation SC 200. A similar situation exists with the MIPS processors, which were also originally developed for other application areas.

Although 32-bit processors take up significantly more die space than 8-bit processors using the same technology, due to their wider buses and more complex internal structures, they will be used in increasing numbers in future smart card applications. The processing power that they offer is indispensable for these applications, so the disadvantages of greater power consumption and increased chip area can be accepted as the price of progress. Of course, 8-bit processors will not die out in the foreseeable future, since they provide a solid basis for inexpensive chips at the lower end of the performance scale.

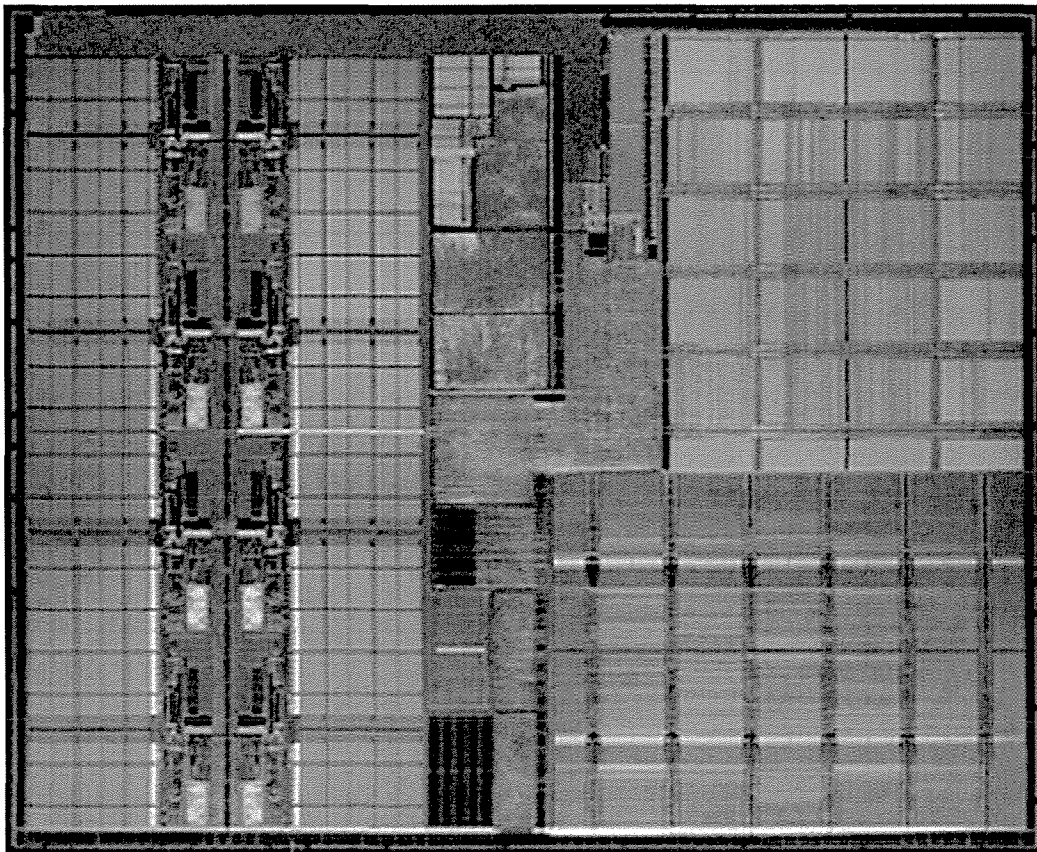


Figure 3.46 Bond-out version of an SLE88CX720P 32-bit smart card microcontroller without final chip shielding. This microcontroller was fabricated in 0.22- μm technology and has 240 kB of ROM, 80 kB of EEPROM and 8 kB of RAM (*Photo: Infineon Technologies*)

3.4.2 Memory types

Besides the processor, the most important components of a microcontroller are various types of memory, which serve to store program code and data. Since smart card microcontrollers must be complete computers, they exhibit a characteristic division of memory into RAM, ROM and EEPROM. The exact division depends very strongly on the chip's ultimate application area. In any case, an effort is always made to keep the RAM and EEPROM as small as possible, since they require the most space per bit.

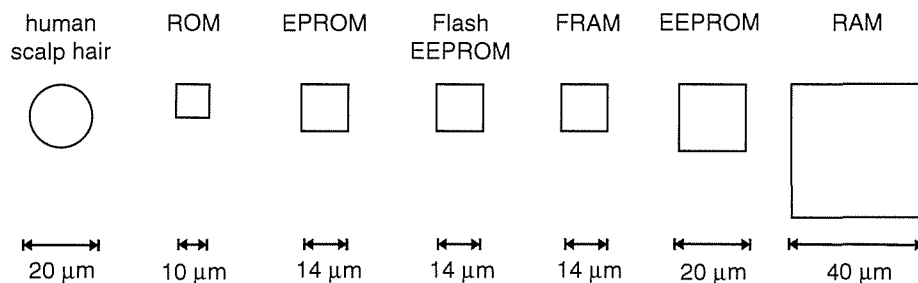


Figure 3.47 Comparison of the die area of a single bit cell for various types of memory. The dimensions shown here are approximate and relate to 0.8-μm technology. For comparison, the diameter of the first planar transistor in 1959 was 764 μm [Buchmann 96, Stix 96], the diameter of the dot at the end of each sentence in this book is 400 μm, the resolution limit of the human eye is 40 μm, the size of a bacterium is 0.4–2 μm and the size of a DNA double helix is 0.1 μm

In the case of multiapplication smart cards, which can manage several applications at the same time, the most commonly used chips have a ROM capacity that is roughly twice as large as that of the EEPROM, in order to provide enough room to store the complex operating system code. For single-application smart cards, microcontrollers are selected whose EEPROM capacity is only slightly larger than the volume of the application data. All variable application data, along with some parts of the operating system, can thus be stored in EEPROM in order to make optimum use of the EEPROM, which takes up a relatively large amount of space on the die and is thus expensive.

Integrating three different types of semiconductor memory into a single silicon die is a technically difficult task requiring a significant number of production steps and exposure masks. The different types of memory also occupy markedly different areas, due to their different structures and operating principles. For example, a RAM cell occupies about four times as much space as an EEPROM cell, which in turn occupies four times as much space as a ROM cell. This is why smart card microcontrollers have so little RAM, with 4 kB of RAM already considered to be large. If you consider that 16 kB of EEPROM or 64 kB of ROM can be put into the same area, you can understand why.

A new type of memory technology for smart cards has become available relatively recently. This is called 'Flash' EEPROM, and it permits write and erase access times that are much shorter than with previously available types of EEPROM. The cell size is approximately half of that of a conventional EEPROM, depending on the particular design.

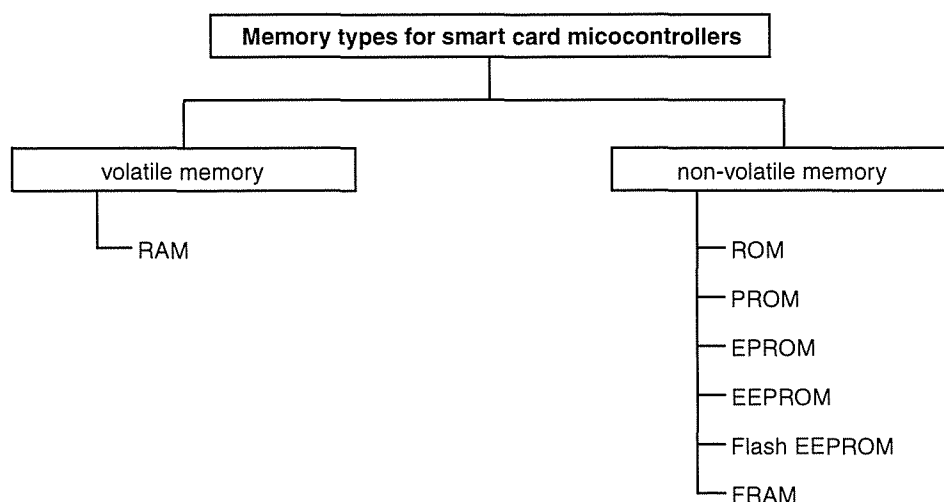


Figure 3.48 Classification chart for smart card microcontroller memories. Generally speaking, PROM and EPROM are no longer used in modern microcontrollers. FRAM is only starting to be used in smart cards

Table 3.4 Memory types used in smart card microcontrollers. For comparison, the area of the dot at the end of each sentence in this book is $125,660 \mu\text{m}^2$

Type of memory	Number of possible write/erase cycles	Write time per memory cell	Typical cell size with 0.8- μm technology
Volatile memory			
RAM	unlimited	$\approx 70 \text{ ns}$	$\cong 1700 \mu\text{m}^2$
Non-volatile memory			
EEPROM	100,000–1,000,000	3–10 ms	$\approx 400 \mu\text{m}^2$
EPROM	1 (not UV-erasable)	$\approx 50 \text{ ms}$	$\approx 200 \mu\text{m}^2$
Flash EEPROM	$\approx 10,000$	$\approx 10 \mu\text{s}$	$\approx 200 \mu\text{m}^2$
FRAM	$\approx 10^{10}$	$\approx 100 \text{ ns}$	$\approx 200 \mu\text{m}^2$
PROM	1	$\approx 100 \text{ ms}$	—
ROM	0	—	$\approx 100 \mu\text{m}^2$

The following three numeric examples illustrate these size relationships:

- A simple laser printer works at a resolution of 600 dpi (dots per inch), which means that the minimum possible dot size is $42.6 \mu\text{m}$. Also, the dot at the end of this sentence has a diameter of $400 \mu\text{m}$. If you wanted to print with a resolution equal to a $0.8\text{-}\mu\text{m}$ structure width, which is still used in semiconductor technology, you need a printer with a resolution of 32,000 dpi!
- High-capacity hard disk drives can store up to 11.6 billion bits per square inch. Under the idealized assumption that each bit occupies a square area, this yields an edge length of $0.24 \mu\text{m}$ for each bit cell. A ROM cell of a smart card microcontroller made using $0.8 \mu\text{m}$ technology requires 1700 times as much area for a single bit!

- With a CD-ROM, the situation is different. In this case, the storage density is 7.3 MB/cm^2 . This corresponds to an edge length of $1.4 \text{ }\mu\text{m}$ for single bit cell, assuming square cells. This is around 80 times less than the area occupied by a ROM cell in $0.8\text{-}\mu\text{m}$ technology. With DVDs (digital versatile disks), a density of 50.5 MB/cm^2 is possible. A single bit thus occupies the area of a square with an edge length of $0.5 \text{ }\mu\text{m}$, which is 400 times smaller than a single ROM bit cell in $0.8\text{-}\mu\text{m}$ technology.

Table 3.5 Typical surface area distribution for a smart card microcontroller

Type of component or memory	Surface area
CPU and NPU	20 %
ROM	10 %
EEPROM	45 %
RAM	15 %
Miscellaneous	10 %

ROM (read-only memory)

As the name implies, this type of memory can only be read and cannot be written. No supply voltage is needed to retain data, since the data are ‘hard-wired’ in the memory.

A smart card’s ROM contains most of the operating system routines, as well as various test and diagnostic functions. These programs are built into the chip by its manufacturer when it is made. This is done by preparing a ROM mask from the program code and then using this mask to ‘burn’ the program into the chip using lithographic processes. In this case the data, which are the same for all chips of a production run, can only be entered into the ROM during manufacturing.

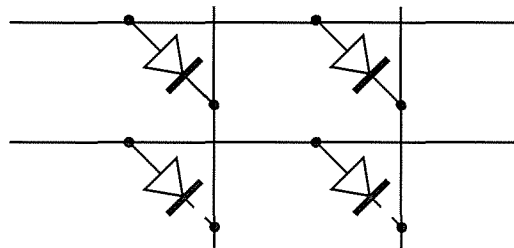


Figure 3.49 Basic functional structure of a ROM

PROM (programmable read-only memory)

PROM is not used in smart card microcontrollers, though it could offer several advantages. In contrast to ROM, PROM need not be programmed during manufacturing, but can be written shortly before the chip is fitted into its module. PROM also does not need any supply voltage

to retain data. The main reason for not using PROM is that programming a PROM requires access to the address, data and control buses. This is precisely what should not be possible with smart cards, because it would allow data to not only be written but also read out. Since the memory holds confidential data, using PROM is strictly prohibited.

EPROM (erasable programmable read-only memory)

EPROM was often used in the early years of smart card technology, since at that time it was the only type of memory that could retain data without a supply voltage and could also be written (although only once per bit). However, since an EPROM can only be erased using UV light, it cannot be erased in a smart card. This is why EPROM no longer has any practical significance.

The only meaningful use for EPROM is to irreversibly store a chip number during semiconductor production, but this can now be realized using a special type of non-erasable EEPROM.

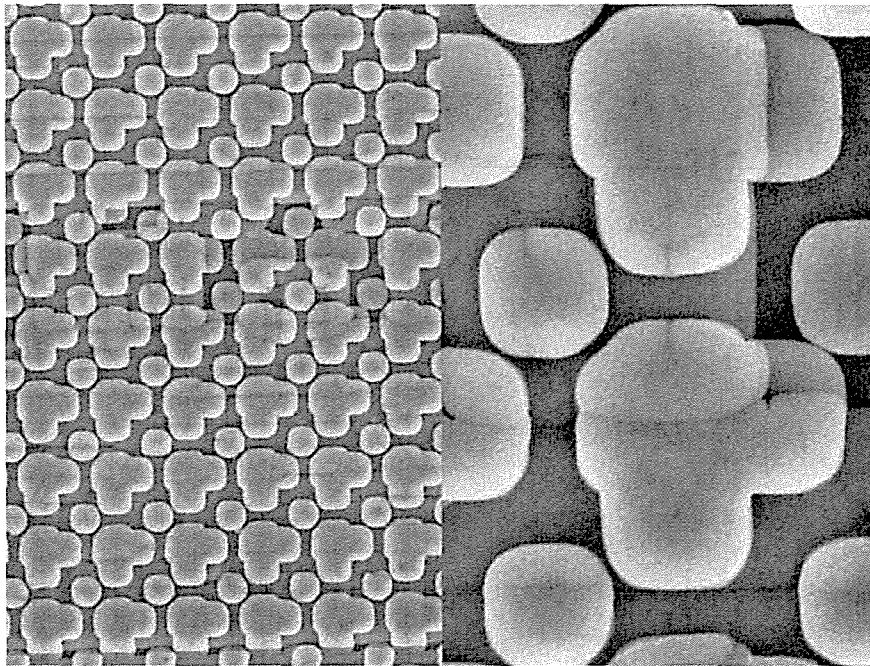


Figure 3.50 Photo of a ROM cell at 1000× enlargement (left) and 11,200× enlargement (right)
(Source: Giesecke & Devrient)

EEPROM (electrically erasable programmable read-only memory)

EEPROM, which is technically more complex than ROM or RAM, is used in smart cards for all data and programs that need to be modified or erased at some time. Functionally, an EEPROM corresponds to the hard disk of a PC, since it retains data in the absence of power and the data can be altered as necessary. EEPROM is thus non-volatile memory.

In principle, an EEPROM cell is a tiny capacitor that can be charged or discharged. The charge state can be interrogated by sensing logic. A charged capacitor represents a logic 1,

while a discharged capacitor represents a logic 0. In order to store one data byte, eight of these small capacitors are needed, along with suitable sensing circuitry.

The erased state of the EEPROM cell is the critical factor with regard to writing to the cell. In most types of EEPROM, the erased state is $^{\circ}1^{\circ}$. An EEPROM has the property that an individual cell can only be programmed from its erased state to its unerased state, which in this example is $^{\circ}0^{\circ}$. If an EEPROM cell is already in the $^{\circ}0^{\circ}$ state, an entire EEPROM page must be erased in order to restore that bit to the $^{\circ}1^{\circ}$ state. The algorithm that is usually used for an EEPROM write routine is described in Listing 3.1.

Listing 3.1 Pseudocode of a routine for writing complete EEPROM pages. If multiple pages or only part of a page is to be written, this routine should be nested in a higher-level routine. A similar procedure should be used if a write retry routine must be called in the event of an error. Here the erased state of the EEPROM is 'FF', and the written state is '00'

UpdateEEPROM: // <i>NewData</i> : data to be written // <i>StoredData</i> : stored data IF (<i>NewData</i> = <i>StoredData</i>) THEN (GOTO UpdateEEPROM.Exit) <i>WorkData</i> := <i>NewData</i> XOR <i>StoredData</i> <i>WorkData</i> := <i>WorkData</i> AND <i>NewData</i> IF (<i>WorkData</i> <> 0) THEN (Erase EEPROM_Page IF (<i>StoredData</i> <> 'FF') THEN (GOTO UpdateEEPROM.Error.Exit)) Write EEPROM_Page with <i>NewData</i> IF (<i>StoredData</i> <> <i>NewData</i>) THEN (GOTO UpdateEEPROM.Error.Exit)	Entry point for writing data to an EEPROM page If the data already stored in the EEPROM page are the same as the new data, exit the function. Following the XOR operation, the differences between the stored data and the new data can be seen as set bits in the variable <i>WorkData</i> . The AND operation causes the variable <i>WorkData</i> to be non-zero if the EEPROM page must be erased before the write process. If the variable <i>WorkData</i> is non-zero, the EEPROM page must be erased before the write operation. After this operation, a test is made to see if the EEPROM page was successfully erased. The EEPROM page can now be written. Afterwards, a check is made to see whether the data were successfully written to the EEPROM.
Update EEPROM.Exit: RETURN	The function has completed successfully.
Update EEPROM.Error.Exit: RETURN	An error occurred during execution of the function.

Figure 3.52 shows the cross-section of an EEPROM cell. The actual structure is somewhat more complicated, but this simplified diagram is a very useful aid to comprehension.

In order to understand how an EEPROM cell works, you need to understand its semiconductor background. In its simplest form, an EEPROM cell is essentially a modified field-effect transistor (MOSFET) built on top of a silicon substrate. A MOSFET is formed by first creating a source and a drain in the substrate and then placing a control gate between them. The current

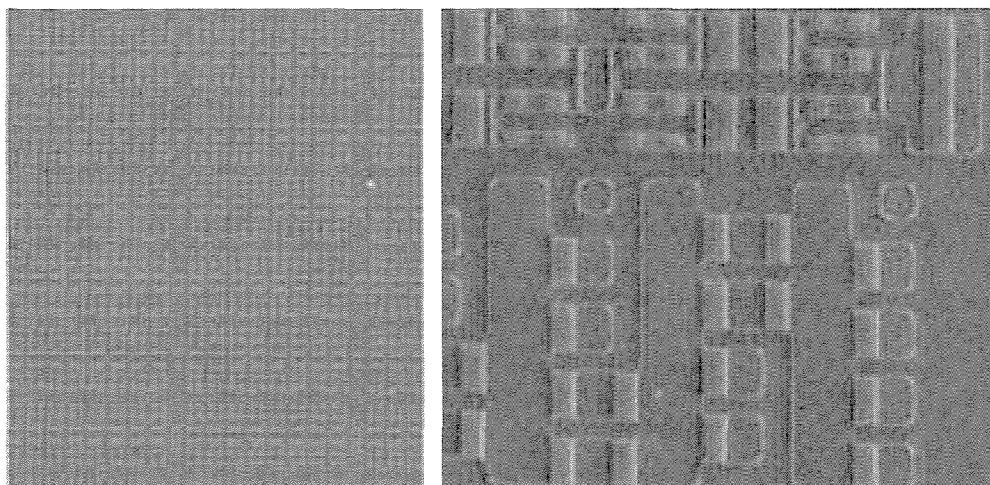


Figure 3.51 Photo of an EEPROM cell at 1000× enlargement (left) and 4000× enlargement (right) (Source: Giesecke & Devrient)

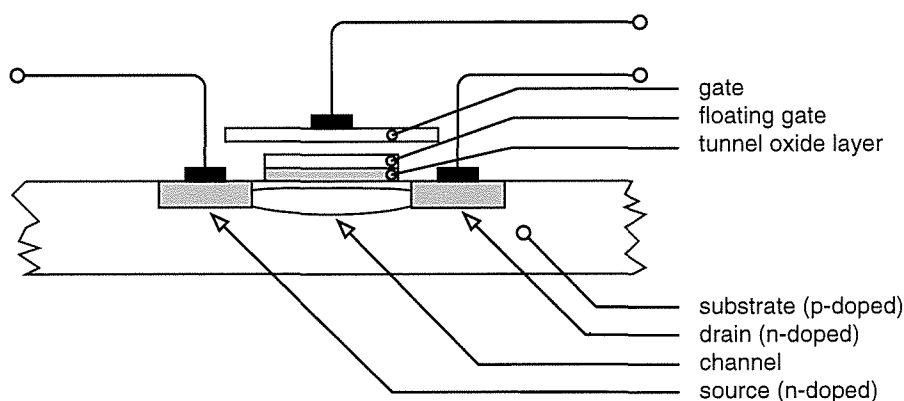


Figure 3.52 Cross-section of the semiconductor structure of an EEPROM cell

flowing from the source to the drain can be controlled by applying a potential to this gate. As long as no potential is present on the gate, no current can flow, since there are two diode junctions (n–p and p–n) between the source and the drain. If a positive potential is applied to the gate, electrons are drawn towards it from the substrate, forming an electrically conducting channel between the source and the drain. The FET is then conductive, and a current can flow.

In an EEPROM cell, an additional ‘floating’ gate is located between the control gate and the substrate. It is not connected to any external voltage source, and the separation between it and the substrate is very small, on the order of 10 nm. The floating gate can be charged or discharged via the substrate using the tunnel effect (Fowler–Nordheim effect), which allows charge carriers to penetrate thin oxide layers that act as insulators. This requires a sufficiently large potential difference across this oxide layer, which is called the tunnel-oxide layer. Current flow from the source to the drain is controlled by the charge on the floating gate. This means that the state of this gate can be interpreted as a logic 0 or a logic 1 according to whether a current can flow through the gate.

To charge the floating gate, a high positive voltage is applied to the control gate. This creates a large potential difference between the substrate and the floating gate, which in turn causes electrons to tunnel through the oxide layer to the floating gate, with a current that can be measured in picoamperes. The negative charge on the floating gate produces a high threshold voltage between the source and the drain, which means that the field-effect transistor is blocked. No current can flow between the source and the drain. Storing electrons in the floating gate is thus equivalent to storing data.

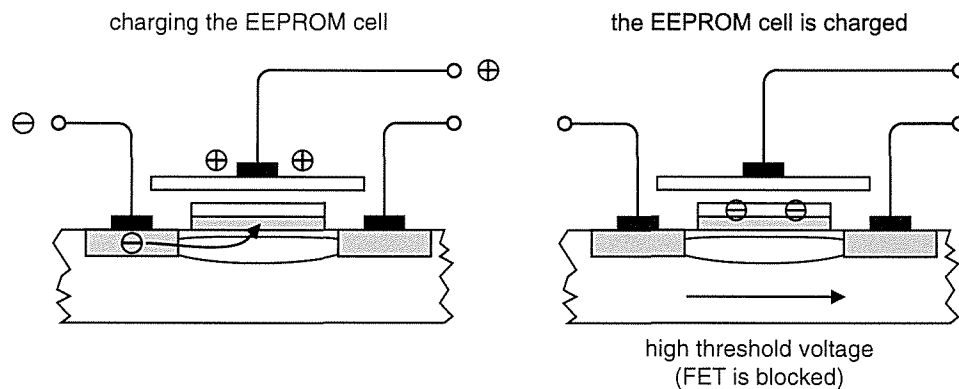


Figure 3.53 Charging an EEPROM cell

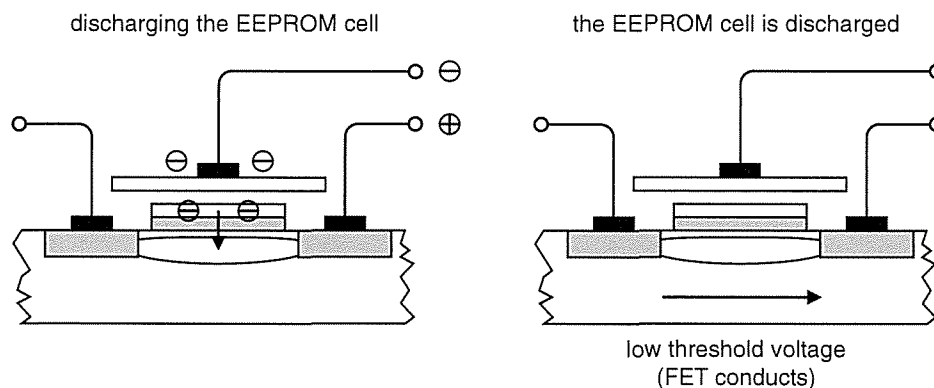


Figure 3.54 Discharging an EEPROM cell

The potential needed to charge the EEPROM cell is about 17 V at the control gate, which is reduced to about 12 V at the floating gate by capacitive coupling. However, since smart card microcontrollers work with a supply voltage of only 1.8–5 V, a charge pump is needed to produce the necessary voltage. In principle, the charge pump is a cascaded voltage-multiplier circuit. It generates an output voltage of about 25 V from the low input voltage, which yields a voltage close to the necessary level of 17 V after stabilization. Depending on the structure of the cell, charging an EEPROM cell requires from 2 to 10 ms per memory page (1–32 bytes).

To erase an EEPROM cell, a negative voltage is applied to the control gate. This causes the electrons to leave the floating gate and return to the substrate. The EEPROM cell is then

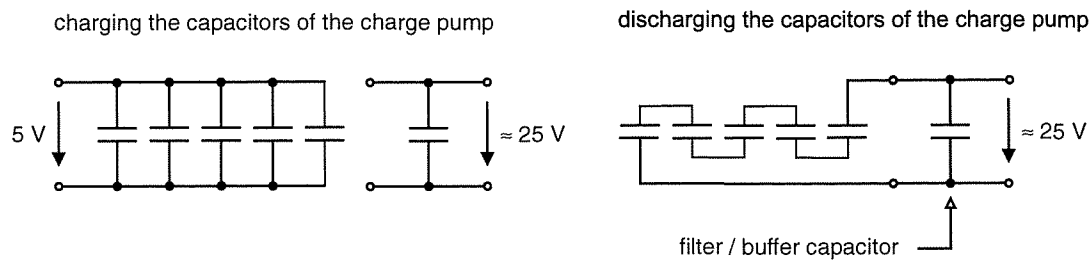


Figure 3.55 This schematic diagram shows the operating principle of a charge pump circuit during charging (left) and discharging (right). These processes are repeated at a high frequency, causing the charge pump to produce a slightly pulsating DC voltage at its output.

discharged and the threshold voltage between the source and the drain is low, so the FET conducts.

The floating gate can also be discharged by heat or energetic radiation (such as X-rays or UV light), which causes it to return to its 'secure' state. This state is of fundamental significance in the design of smart card operating systems, since security barriers can be breached by deliberately altering ambient conditions if the secure state of the EEPROM is not used to store critical data. Depending on the technical implementation of an EEPROM cell, the secure state can correspond to a logic 0 or a logic 1. This is specific to each type of smart card microcontroller, and it should be confirmed with the manufacturer if necessary.

EEPROM is one of the few types of semiconductor memory having a limited number of access cycles. It can be read any number of times, but it can be programmed only a limited number of times. The reason for this limitation can be found in its semiconductor structure. The life expectancy of an EEPROM depends strongly on the nature, thickness and quality of the tunnel-oxide layer between the floating gate and the substrate. Since this layer must be produced very early in the fabrication process, it is exposed to strong thermal stresses in subsequent fabrication steps. This may cause damage to the oxide layer, which in turn affects the useful life of the EEPROM cell. During fabrication, and every time the cell is written, the tunnel-oxide layer absorbs electrons that are not subsequently released. These 'trapped' electrons are located close to the channel between the source and the drain, and once they reach a certain number they have a stronger effect on the threshold potential than the charge stored in the floating gate. When this happens, the EEPROM cell has reached the end of its useful life. Although it can still be written, the charge on the floating gate has only a minimal effect on the characteristics of the channel between the source and the drain, so the threshold potential always remains the same. The number of possible write/erase cycles varies greatly, depending on structural details. Typical values range from 100,000 to 1,000,000 cycles over the entire range of operating temperature and voltage. At room temperature and using an optimum supply voltage, values that are 10 to 50 times greater can be achieved.

When an EEPROM cell is approaching the end of its life, its data retention time decreases. The retention time can range from hours to minutes or even seconds. The more exhausted the EEPROM becomes, i.e., the more electrons that have been absorbed by the tunnel oxide layer, the shorter is the retention time.

A charged floating gate loses charge over time, due to insulation losses and quantum-mechanical effects. The time required for this to become noticeable can range from 10 to

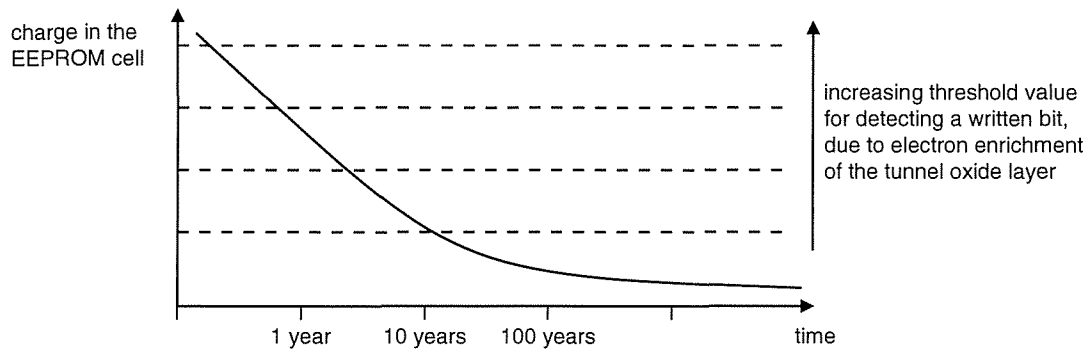


Figure 3.56 Displacement of the discharge curve of an EEPROM cell as a function of the number of executed program/erase cycles

100 years. In this regard, it is interesting to note that a charged floating gate holds 100,000 to 1,000,000 electrons, depending on the implementation. Currently, all semiconductor manufacturers guarantee data retention for 10 years. In order to increase this value, the contents of EEPROM cells can be periodically refreshed by reprogramming. However, this is only worthwhile when the data must be stored for a long time.

Flash EEPROM (Flash electrically erasable programmable read-only memory)

Flash EEPROM, which is often simply called ‘Flash memory’, shares the property of non-volatility with regular EEPROM. This means that it retains data in the absence of a supply voltage. It is very similar to EEPROM in its construction and operation. The basic difference between a Flash EEPROM and a normal EEPROM is in the writing process, which is based on hot-electron injection instead of the Fowler–Nordheim (tunneling) effect. ‘Hot’ electrons are fast electrons produced by a high potential difference between the source and the drain. Some of these electrons penetrate the tunnel-oxide layer, due to the influence of a positively charged control gate, and are stored in the floating gate. This reduces the writing time to around 10 μ s, which is a considerable improvement on the value of 2–10 ms for a regular EEPROM. The name ‘Flash’ comes from this extremely short programming time. Another advantage is that the programming voltage is only 12 V, compared with 17 V for EEPROMs.

There are several smart card microcontrollers with Flash EEPROM, which is primarily used in smart card microcontrollers as a replacement for mask-programmed ROM. Using a microcontroller with Flash EEPROM can reduce the development time of a smart card project by several months, since this eliminates the need to generate ROM masks.

Unfortunately, it is extremely difficult to make semiconductor devices having EEPROM and Flash EEPROM on the same chip. Consequently, in practice a microcontroller with Flash EEPROM usually does not contain any regular EEPROM. Instead, the EEPROM is replaced by a Flash EEPROM of around 8 kB, which has the smallest possible page size in order to minimize the impact on the smart card operating system. The page size of the Flash memory used to replace the ROM is generally significantly larger (e.g., 64–128 bytes), since the routines stored in this memory are written only rarely. When the chip is fabricated, a boot loader is

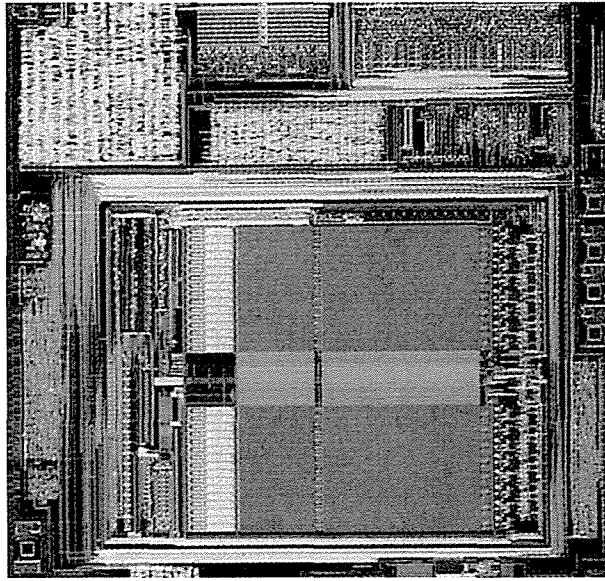


Figure 3.57 An AT89SC168 smart card microcontroller with Flash EEPROM. The functional components at the top are (from left to right) the logic unit, RAM and CPU. The EEPROM charge pump and Flash EEPROM can be seen at the bottom (from left to right) (*Source: Atmel*)

stored in a small ROM to allow the smart card manufacturer to load program code and data into the Flash EEPROM.

Current Flash EEPROM cells have a guaranteed data retention period of at least 10 years, at least 100,000 write/erase cycles and typical page sizes of 8–128 bytes.

There are a few isolated smart card microcontrollers that have unusually large memories, frequently on the order of 1–2 MB. They are always fabricated using Flash memories with page sizes of up to 64 kB. This yields significant area savings with regard to the address and control lines, so memories of this size can be realized in chips having the maximum possible area of 25 mm².

FRAM (ferroelectric random-access memory)

FRAM is a new development in semiconductor technology. Despite its name, FRAM is not volatile like RAM, but instead retains its content without a supply voltage. This type of memory exploits the properties of ferroelectric materials in order to store data. Its cell structure is similar to that of EEPROM, but with a ferroelectric material located between the control gate and the floating gate.

FRAM is potentially ideal for smart card memory, since it has very desirable properties as a data storage medium. Only 5 V is needed for programming, the programming time is around 100 ns and the maximum number of programming cycles is around one trillion. The integration density is similar to that of Flash EEPROM. However, FRAM has two disadvantages. The first is a limited number of read cycles, which makes a type of refresh cycle necessary. The second, which is more significant, is that producing FRAMs involves processing steps that

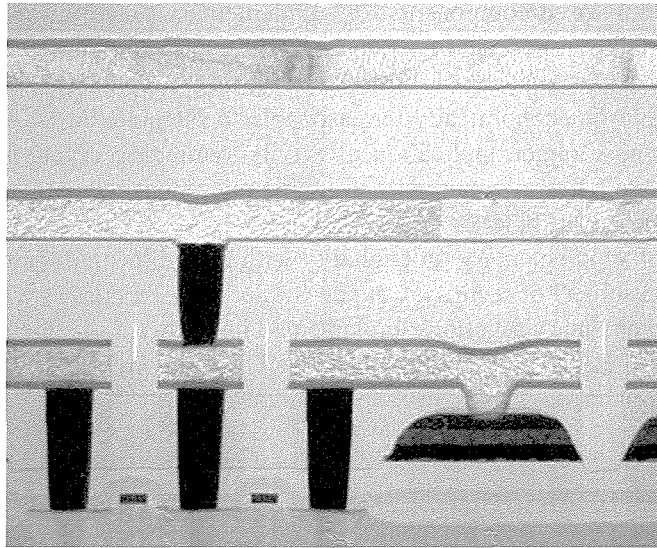


Figure 3.58 Cross-section of a FRAM cell in 0.35- μm technology. The light horizontal bands are aluminum metallization layers, and the dark vertical bars are interconnections ('vias') between the layers. The trapezoidal horizontal area at the lower right is the actual FRAM cell. The width of the cell is approximately 1.5 μm (*Source: Fujitsu*)

are difficult to master. Up to now, little effort has been made to use this technology in smart card microcontrollers. However, this could change in a few years, since FRAM technology possesses all the features needed to allow it to completely supplant EEPROMs, which are presently used almost exclusively.

RAM (random-access memory)

In smart cards, RAM is the memory used to hold data that are stored or altered during a session. The number of accesses is unlimited. RAM needs a power supply in order to operate. If power is switched off or fails temporarily, the content of the RAM is undefined.

A RAM cell consists of several transistors, connected such that they work as a bistable multivibrator. The state of this multivibrator represents the stored value of one bit in the RAM. The RAM used in smart cards is static (SRAM), which means that its contents do not have to be periodically refreshed. It is thus not dependent on an external clock, in contrast to dynamic RAM (DRAM). It is important for the RAM to be static, since it must be possible to stop the clock signal to a smart card. With dynamic RAM, this would cause the stored information to be lost.

3.4.3 Supplementary hardware

There are some requirements specific to smart cards that cannot be fully satisfied using software and thus must be satisfied by supplementary hardware, since they cannot be satisfied using the hardware of conventional microcontrollers. Consequently, the various manufacturers of smart card microcontrollers offer a wide range of supplementary functions in the form of on-chip hardware.

The most commonly used components for supplementary functions are described below. These components do not necessarily have to all be present in any particular microcontroller. The components that are present depend strongly on the target application, among other things. For example, it would be economically unreasonable to integrate an RSA coprocessor into a microcontroller whose target application uses only symmetric cryptographic algorithms. Nevertheless, there are a few commercially available microcontrollers that include nearly all of the components described below.

Another aspect of supplementary functionality with regard to smart card microcontrollers relates to the general subject of security. Chapter 8, 'Security Techniques', contains extensive descriptions of supplementary functions implemented in hardware that are primarily intended to counter possible attacks. Consequently, here we describe only those components whose primary purpose is not enhancing security against attacks.

Hardware-based data transmission (UARTs)

The only communications between a smart card and the outside world take place via a bi-directional serial interface. Originally, data transmission and reception via this interface were controlled exclusively by operating system software, without any hardware support. This requires very complex software, and it creates additional potential sources of software errors. However, the main problem is that the speed of software-based data transmission is limited, since the speed of the processor is limited. With current processors, the upper limit is represented by a divider value (clock rate conversion factor) of around 30, which yields a data transmission rate of approximately 115 kbit/s with a 3.5-MHz clock.

If higher communication speeds are desired or required, it is necessary to use either internal clock multiplication or a UART (universal asynchronous receiver-transmitter) component. As the name suggests, such a component is a general-purpose component for transmitting and receiving data independent of the processor. It is not limited by the speed of the processor, nor does it need software for communicating at the byte level. Of course, the upper layers of the data transmission protocol still must be present in the smart card in the form of software, but the lowest layer is implemented in hardware in the UART.

Current UARTs can generally work with divider values smaller than 372, in line with ISO/IEC 7816-3, and some of them can transmit and receive data with divider values as small as 1. There is a wide range of implementations of this function. Some UARTs can transmit and receive only single bytes, and only support byte retransmission according to the $T = 1$ protocol in the event of a transmission error. With such UARTs, all the processor has to do is to supply the necessary data to the UART on time and read it from the UART on time. Reception of a complete byte can be signaled to the processor by a flag or interrupt. All more advanced UARTs can transmit or receive multiple bytes in succession. The highest level of technical capability is presently provided by UARTs that can directly transmit data from the RAM or store received data in the RAM using direct memory access (DMA), without the intervention of the processor and in parallel with the other activities of the processor.

It has been technically feasible to implement UARTs in smart card microcontrollers since the origin of smart cards, but until the end of the 1990s, transmission and reception routines implemented in ROM software required less physical area on the silicon than a functionally equivalent UART component. Since the total surface area is a decisive cost factor for smart card

microcontrollers, for a long time nearly all semiconductor manufacturers rejected the hardware approach. However, conditions have changed with increasing integration density. UARTs with a wide range of capabilities are now standard components of all smart card microcontrollers.

Many new types of microcontrollers also allow a USB interface to be placed on the chip as an optional component, in addition to a UART. With such an interface, it would be possible to exchange data with a terminal using the USB protocol with hardware support. Unfortunately, up to now it is effectively not possible to use this USB hardware extension, since USB on smart cards is not yet covered by any standard, which means that it is impossible to guarantee the mutual compatibility of smart cards and terminals.

Timers and watchdogs

Timers in smart card microcontrollers are connected to the internal processor clock or UART clock (which counts etu's) via a configurable divider. They usually have a counting range of 16 or (more rarely) 32 bits. Using a timer, the number of clock pulses from the Start command to the End command can be measured without involving the processor. Most timers can also be used in the reloadable mode, in which they count down from a predefined value and trigger an interrupt when the count reaches zero, after which the counter is automatically reset to the initial value and continues counting.

A watchdog is also often present in the microcontroller. In principle, a watchdog is a timer that must be regularly reset by an explicit processor instruction, in order to prevent it from timing out after a present interval and triggering a reset. A watchdog allows the processor to be reset to a defined state after a definable maximum interval if it becomes trapped in an endless loop. The primary typical application for watchdogs is in the autonomous controller environment, where they are highly useful. However, they are not of much use for smart cards, partly because the software is (hopefully) extremely reliable and partly because the terminal can always interrupt the processor if it lands in an endless loop. Consequently, watchdogs are generally not used in smart card microcontrollers.

Internal clock multiplication and generation

The demands on the processing power of smart cards are constantly increasing. This applies to the processor as well as components that support cryptographic algorithms. One way to meet these demands is to simply increase the frequency of the clock applied to the microcontroller, since processing power is proportional to the clock rate. Doubling the clock rate thus doubles the performance of the processor. However, for reasons of compatibility with applicable standards, it is generally not possible to increase clock rate above 5 MHz.

To get around this restriction, the internal clock frequency of the microcontroller can be increased by using a clock multiplier. This is technically realized using a phase-locked loop (PLL) circuit, which is a well-proven technique, or an RC oscillator. For instance, a smart card connected to an external 5-MHz clock can be operated internally at 20 MHz, which provides significant benefits with regard to computation times for complex cryptographic algorithms or running a Java virtual machine.

Nevertheless, when clock multiplication or an internal clock generator is used, it must be remembered that a higher clock rate causes a proportional increase in current consumption. As a rule, the relationship between clock frequency and current consumption is linear, which means that quadrupling the clock frequency (for example) also quadruples the current consumption. Particularly with battery-operated terminals, increased current consumption is not desirable.

An elegant solution to this problem is provided by ‘intelligent power management’ in the microcontroller, which involves communicating the maximum allowable current consumption to the control logic of the PLL. This logic then adjusts the PLL to operate in a frequency range that avoids exceeding the prescribed maximum current consumption, without any involvement by the processor. For instance, if the power-hungry NPU is switched into the processing loop, the internal clock frequency will be automatically reduced to prevent the current consumption from rising above the permissible value. Unfortunately, there is a small difficulty with this solution, which is that the specifications for smart cards for GSM and UMTS telecommunications (presently) prohibit the use of free-running oscillators in smart card microcontrollers. This prohibition is based on fear of possible interference with the other circuitry in the mobile telephone. As long as these portions of the specifications continue to exist, it is not possible to use either a continuously adjustable internal clock frequency or an oscillator that is completely independent of the applied clock signal. However, these specifications do allow clock rate multipliers to be used, as long as the internal and external clock rates have a fixed relationship governed by predefined multiplication factors.

Processor speed is not the only bottleneck in smart cards. Data transmission rates, which are specified in the standards, and EEPROM write and erase times do not benefit from increased clock rates. This somewhat limits the advantage of increasing the clock rate. Nevertheless, it can be highly beneficial to use a smart card with an elevated internal clock rate for certain applications, particularly considering that the amount of additional circuitry (and thus area) required on the chip is small. For this reason, nearly all new types of smart card microcontrollers have internal clock multiplication capability.

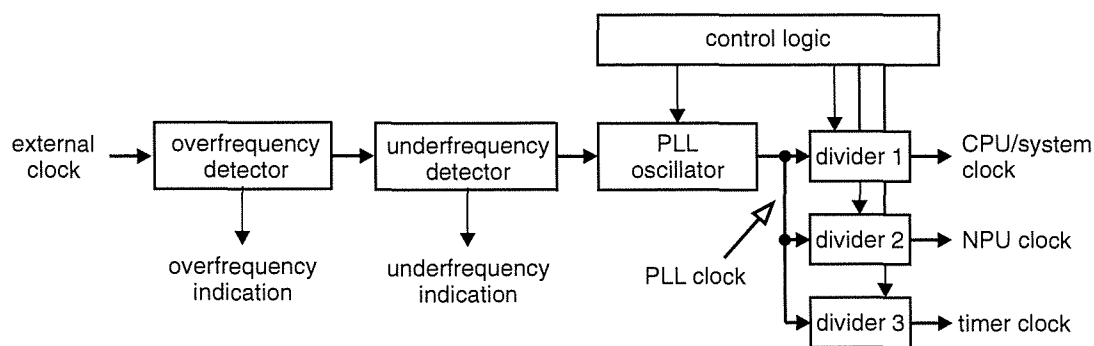


Figure 3.59 Block diagram of a possible internal clock multiplication circuit using a PLL oscillator followed by a divider to supply clock signals to the various microcontroller components. The clock for the NPU is often taken directly from the PLL without any division. If there are several timers in the microcontroller, each one usually has its own individually configurable predivider

DMA (direct memory access)

DMA components have been used for a long time in the PC realm. DMA makes it possible to copy or exchange data between two or more memory regions at high speed, independent of the processor and in part in parallel with the other activities of the processor. It is often also possible to independently fill a certain memory region with a predefined value. The main effect of a DMA unit is to offload the processor and thus allow certain routines to be fashioned more simply. Up to now, high-performance DMA components have been sporadically available in smart card microcontrollers.

Hardware-based memory management, firewalls and memory management units (MMUs)

The latest smart card operating systems allow executable machine code to be downloaded directly to the card.⁴ This code, which can then be run using a special command, can be used for purposes such as executing a cryptographic function only known to the card issuer. However, it is in principle not possible to prevent such downloaded code from including a function for reading out secret data from the memory. Operating system manufacturers have been very careful to maintain the confidentiality of their system architectures and program code. The same is also true of secret keys and algorithms in various applications in the card. The public availability of such confidential information would have fatal consequences for an application provider. One administrative solution is to have every new program tested by an independent organization. However, even this cannot guarantee complete security, since a program that is not the same as the one that was tested could later be substituted for the certified program, or the program might be so secret that nobody other than the application provider is allowed to know about it.

One acceptable solution to this impasse is to equip the smart card microcontroller with a memory management unit (MMU). Such a unit monitors the memory boundaries of the current application program while it is running. The permitted memory region is defined by an operating system routine before the application is called, and it cannot be altered by the application program while it is running. This ensures that the application is completely encapsulated and cannot access memory areas forbidden to it. The barriers formed in this manner are called 'firewalls', in analogy to walls used for fire protection in buildings. If an application attempts to access another memory region from within a region demarcated by firewalls, it will fundamentally be prevented from doing so, and in addition any such attempt usually triggers an interrupt so the violation can be immediately detected.

Presently, very few smart card microcontrollers have MMUs, although they have been used for years in many other areas. Nonetheless, the importance of this supplementary hardware will greatly increase in the future, since it is the only practical way to securely isolate several applications sharing a single smart card.

Another aspect of MMUs is their ability to relocate physically addressable memory regions to any desired location within the logical memory space of the processor. To a certain extent,

⁴ See also Section 5.10, 'Smart Card Operating Systems with Downloadable Program Code'

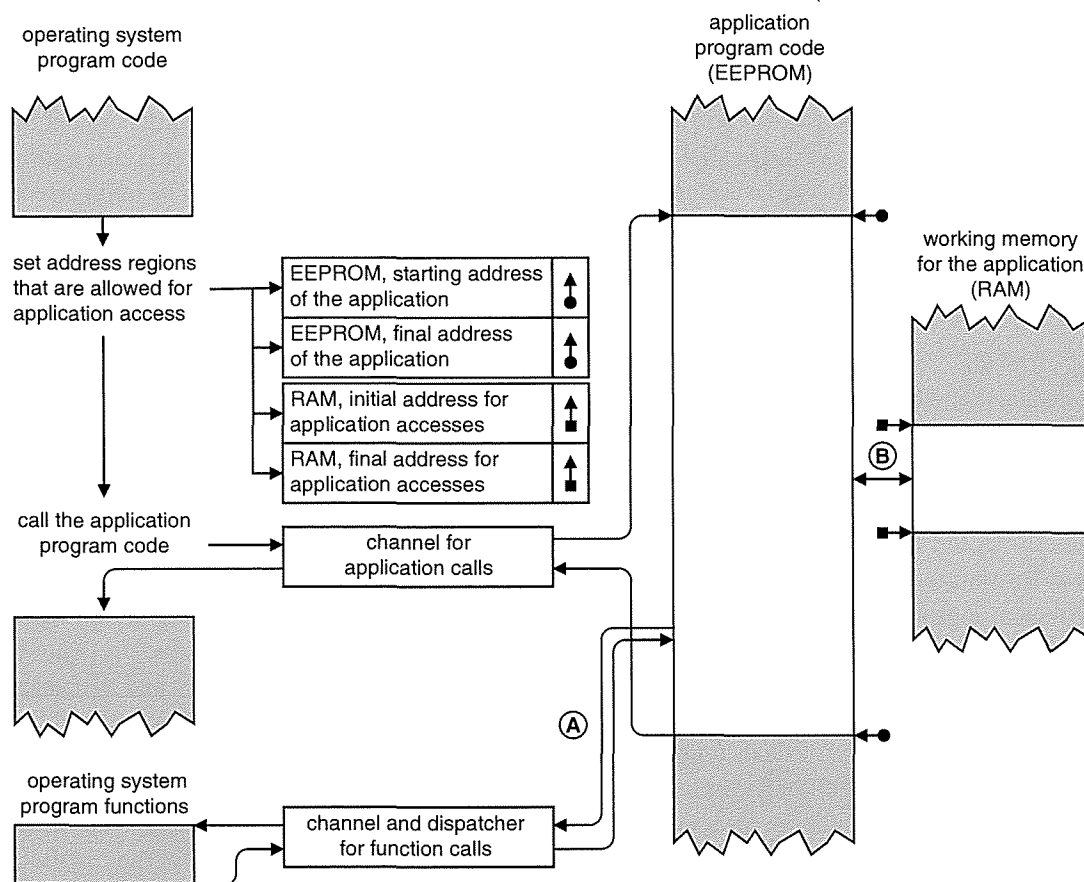


Figure 3.60 Schematic representation of the operating principles of a hardware-based memory management unit (MMU) in a smart card microcontroller. Process 'A' shows a call to an operating-system function that is channeled via the MMU and controlled by a task dispatcher. 'B' is an example of a write-read access to an application memory area demarcated by the MMU

this considerably simplifies the memory management function of the smart card operating system, as well as making it possible to enforce strict isolation of applications with regard to memory space. Furthermore, if downloadable native code is used, the MMU can be used to relocate it to a suitable memory area, thus eliminating the need to use the operating system to manually relocate the executable code.

There is a critical factor that must be considered when using an MMU in a smart card operating system. With the current state of the technology, all MMUs used in various types of microcontrollers are specifically designed for the microcontroller in question. Although this allows the operation and space requirements of the MMU to be optimized, it comes at the price of portability of the object code. In practice, the particular type of MMU that is used has significantly greater consequences for the operating system than the type of processor that is used. Consequently, MMUs are used only very reluctantly in combination with smart card operating systems for large-scale applications, which must of necessity support several different hardware platforms.

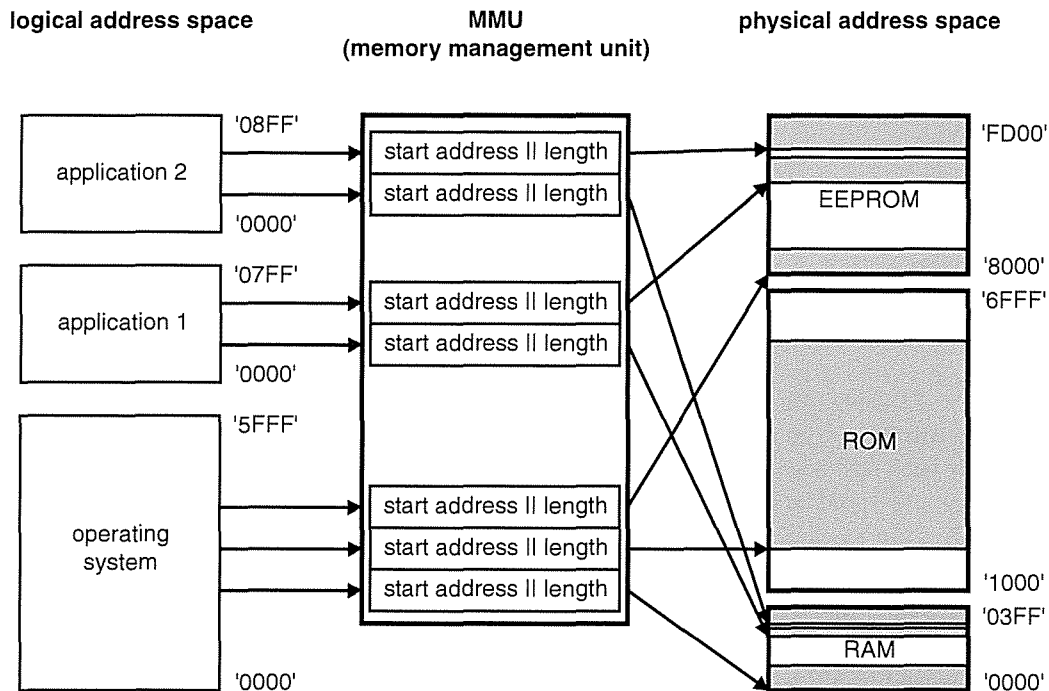


Figure 3.61 Schematic representation of the operating principles of hardware-based memory management (MMU) in a smart card microcontroller with regard to the arrangement of logical and physical address spaces. This example shows an operating system and two applications that share the physically available memory via the MMU. For each of these software components, the MMU translates its physical address space into a logical address space starting at '0000'

CRC (cyclic redundancy check) calculation unit

CRC codes are still frequently used to secure data or programs by means of an error detection code. Calculating a CRC in software is relatively slow, due to the large number of bit manipulations required, and the calculation can be readily implemented in hardware on the silicon of the microcontroller. For this reason, there are microcontrollers for smart cards that have hardware-based CRC calculation units. Naturally, with such units it must be possible to select the usual generator polynomials and seed values.

Random number generator (RNG)

Random numbers are frequently needed in smart cards for generating keys and authenticating smart cards and terminals. For reasons of security, they should be genuine random numbers rather than pseudo-random numbers, as are commonly produced by typical software-based random number generators. All new smart card microcontrollers have hardware random number generators that produce true random numbers.

However, the quality of the numbers produced by such generators must be immune to being adversely affected by external influences, such as temperature or supply voltage. The hardware

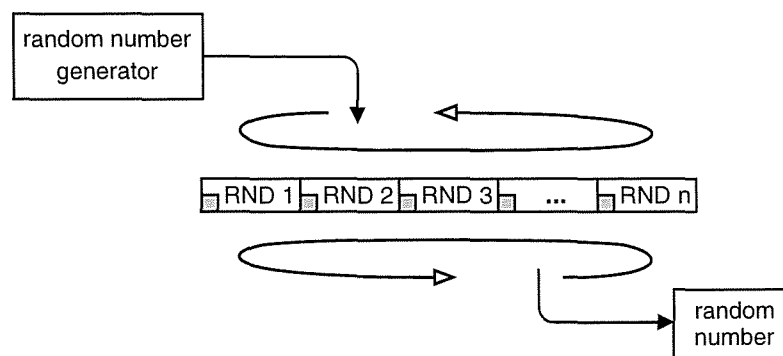


Figure 3.62 Example of a random number generator whose outputs are constantly written to a ring buffer, from which they can be requested as necessary. The gray rectangles mark random numbers that have already been read once and cannot be requested again. This sort of buffer arrangement is used in some types of low-speed random number generators

may use such external influences to assist in generating random numbers, but it must not be possible to predict the generated random numbers by purposefully manipulating one or more of these parameters.

This is very difficult to implement in silicon, so a different approach is taken. The random number generator takes various logic states of the processor, such as the clock signal and the contents of the memory, and applies them to a linear feedback shift register (LFSR) clocked by a signal that is also generated using several different parameters. In some cases, this clock can have a frequency several times that of the processor. If the CPU reads the content of this random number generator, it obtains a relatively good random number that cannot be ascertained from outside in a deterministic manner. The quality of the random number so obtained can be improved by supplementary procedures and algorithms. However, what is important here is that the hardware-based random number generator must basically provide good random numbers that can withstand the usual tests⁵ (e.g., FIPS 140–2).

Java accelerator

Within only two years, Java Card has established itself as an industry standard for executable program code in smart cards. However, since the Java VM must interpret the bytecode rather than directly execute it, there is an unavoidable loss of execution speed compared with native machine instructions, which can be directly executed by the processor. However, the widespread use of Java in smart cards makes it attractive for semiconductor manufacturers to devise remedies for this processing speed problem. Presently, two different approaches are being pursued.

In the first approach, large portions of the Java VM are incorporated into the smart card microcontroller as dedicated hardware components that supplement the actual processor. This technique thus goes in the direction of picoJava, which means in the direction of a real IC that

⁵ The quality of random numbers is treated in overview in Section 4.10.2, ‘Testing random numbers’

can directly process Java bytecode. This solution has two drawbacks, which are that the Java processor takes up additional space, besides that occupied by the regular processor, and that a full implementation of a Java VM is relatively costly. The advantage of this solution is its high execution speed.

In the second approach, the instruction set of the processor is extended to include typical Java machine instructions. This allows bytecodes supplied by the software VM to be immediately processed by the extended processor. This variant is implemented using a processor lookup table containing CPU microinstruction sequences corresponding to the bytecodes to be emulated. The advantage of this solution relative to the first one is that it requires less additional space on the chip, although its execution speed is somewhat lower.

Coprocessors for symmetric cryptographic algorithms

Up to now, DES has been used as the standard cryptographic algorithm for financial transaction systems and telecommunications applications. This large market potential made it worthwhile for semiconductor manufacturers to fit smart card microcontrollers with their own DES calculation units. In principle, this is not particularly difficult, since DES was originally designed to be primarily implemented in hardware. The largest problem in marketing DES calculation units in microcontrollers is not technical, but instead relates to export restrictions, since in many countries components with fast, hardware-based DES encryption are subject to a variety of export regulations.

The advantages of DES calculation units for smart card microcontrollers can be clearly seen by examining their performance figures. At 3.5 MHz, they can achieve times on the order of 75 μ s for a simple DES operation and 150 μ s for a triple-DES operation with two keys. The calculation time decreases linearly as the clock rate is increased. Besides this, a DES calculation unit does not require significantly more chip area than that occupied by the ROM code for a software DES implementation, so it does not increase the size of the die.

In the future, besides DES coprocessors there will also be special coprocessors for AES in smart card microcontrollers, usually supporting all three possible key lengths (128, 196 and 256 bytes). This is technically just as feasible as a DES coprocessor, since the AES algorithm is also relatively easy to implement in hardware.

Coprocessors for asymmetric cryptographic algorithms

For calculations in the realm of public-key algorithms, such as RSA and elliptic-curve algorithms, there are specially developed arithmetic units that are placed on the silicon along with the usual functional components of a smart card microcontroller. These arithmetic units are only capable of performing several basic calculations that are necessary for these types of algorithms, namely exponentiation and modulo calculations using large numbers. The speed of these components, which are optimized for these two arithmetic operations, is due to their very broad architectures (up to 140 bits). In their particular application area, some of them can even outperform a powerful PC.

The arithmetic unit is called by the processor, which either passes the data directly or passes a pointer to the data and then issues an instruction to start the processing. After the task has

been completed and the result has been stored in RAM, control of the chip is returned to the processor. In general, these coprocessors can process all key lengths up to 1024 bits for the RSA algorithm, and in the medium term this will increase to 2048 bits. For elliptic curves, the usual capacity is up to 160 bits, with 210 bits to come in the future.

Error detection and correction in EEPROM

The essential limitation on the useful life of a smart card is imposed by the EEPROM, with its technically limited number of possible write/erase cycles. One way to relax this limitation is to use software to calculate error correction codes for certain heavily used regions of EEPROM, so that errors can be corrected. It is also possible to implement error correction codes using hardware circuitry on the chip. In this way, EEPROM errors can be detected and corrected (as long as they are not too extensive) in a manner that is transparent to the software.

Naturally, additional EEPROM is necessary to store the codes. Since good error correction codes take up a relatively large amount of memory, the designer is confronted with a strategic decision: good error detection demands extra memory – up to 50 % of the memory to be protected. What's more, the memory for the error correction mechanism can only be used for this purpose. Although lower performance error correction requires less additional memory, its usefulness is highly questionable.

There are a few microcontrollers on the market that have EEPROM error detection and correction implemented in hardware, but they may require extra memory amounting to as much as half the volume of the memory to be protected to be used for the protection codes. As a result, the amount of EEPROM available to the user may not be particularly large. However, the useful life of an EEPROM secured using this mechanism is several times the usual value.

Chip hardware extensions

If the chip hardware must be extended for a particular reason, considerable expenditures of development effort and costs are required on the part of the manufacturer. There are only two ways to implement customer-specific hardware: it can be built in silicon on the basis of an existing chip family, or it can be built as a two-chip system, with all of the associated drawbacks.

There is an acceptable solution to this problem in the form of a compromise that incorporates elements of both of these options. A chip with the new hardware unit can be glued directly to the existing chip and electrically connected to it by bonding wires. This solution benefits from the fact that most smart card microcontrollers have several I/O ports, and one of these ports can be used to communicate with the additional chip. The thickness of the resulting sandwich construction is not significantly greater than that of a normal chip, since the silicon substrates can be ground away more than usual to make them thinner. A sandwich chip can thus be built into a standard module without additional effort or costs.

This technique is ideal for satisfying customer-specific needs for additional hardware without expensive redesign. An existing chip can be combined with a new unit, which may for instance have a special serial interface for testing the security features of other chips. It is also possible to fit a special ASIC containing a secret cryptographic algorithm into the card.

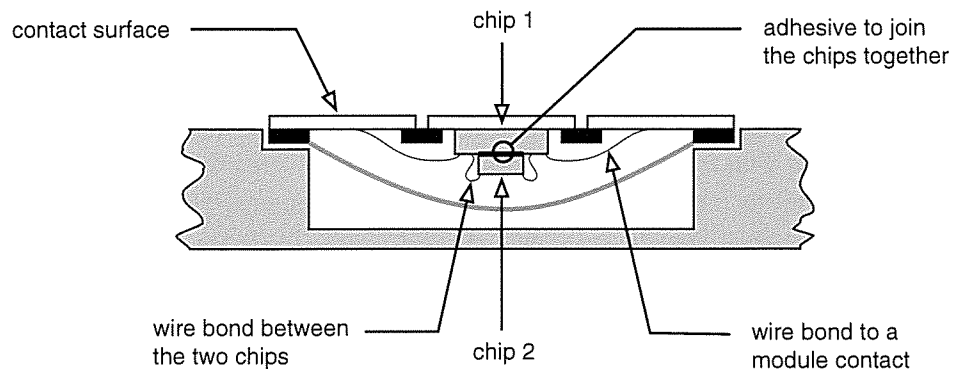


Figure 3.63 Cross section of a chip module containing two different chips electrically interconnected via bonding wires

This method is not cost-effective for large production quantities (in the range of millions of pieces), since in such cases it is worthwhile to develop special chips. However, for small to medium piece counts, sandwich chips are a very effective solution for prototype series or special applications, such as security modules for terminals or smart cards for pay-TV decoders.⁶

Vertical system integration (VSI) and face-to-face

Another technique used to extend chip hardware by combining semiconductor technologies that are incompatible on a single chip is vertical system integration (VSI), in which two or

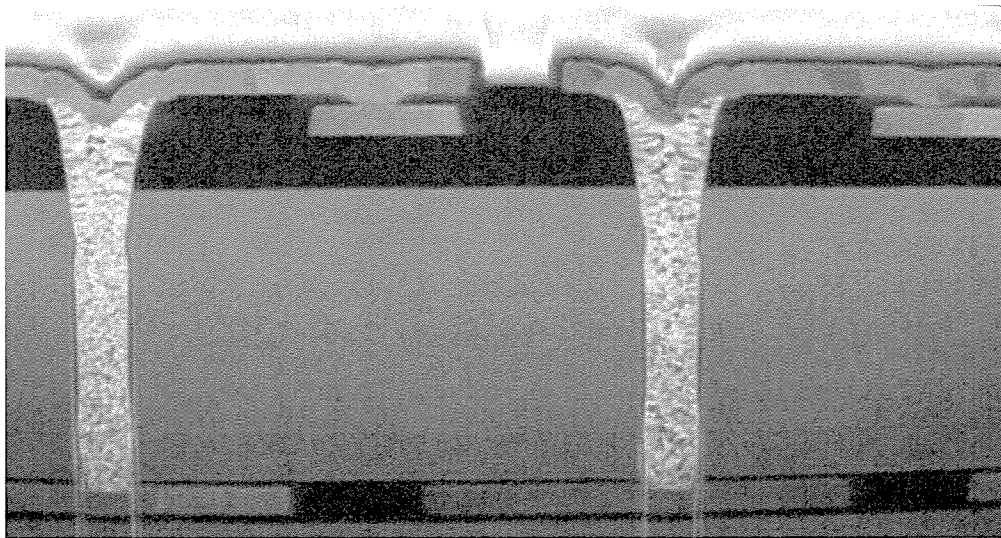


Figure 3.64 Cross-sectional photograph of a VSI stack. The two through contacts between the two stacked dies can be easily recognized (*Source: Giesecke & Devrient*)

⁶ See also [Kuhn]

more dies that have been ground thin are bonded together mechanically to form a stack, with the individual dies also being electrically interconnected by through contacts ('vias') formed using semiconductor fabrication processes. Using VSI, the available chip area can be increased in units of the original area in a very elegant manner. With two stacked dice, twice the original area is available, and with four dice there is four times as much area. It is possible to achieve not only a significant increase in the amount of available memory, but also a considerable improvement in chip security. This is because it is presently effectively impossible to access a chip sandwiched between two other dice using analytical equipment of the type used in the semiconductor industry without destroying the surrounding chips.

A simpler variant of VSI, which in principle can be scaled up as much as desired, is the face-to-face arrangement of two chips. Here the electrical connections are made by extremely precise positioning of the two chips, with their upper surfaces (faces) touching.

VSI and face-to-face chip bonding both allow significantly better extensions of chip hardware to be realized than what can be achieved by interconnecting two chips using wire bonds.

3.5 CONTACT-TYPE CARDS

The main difference between a smart card and other types of cards is the embedded micro-controller. If contacts are used for the power supply and data transmission functions, electrical

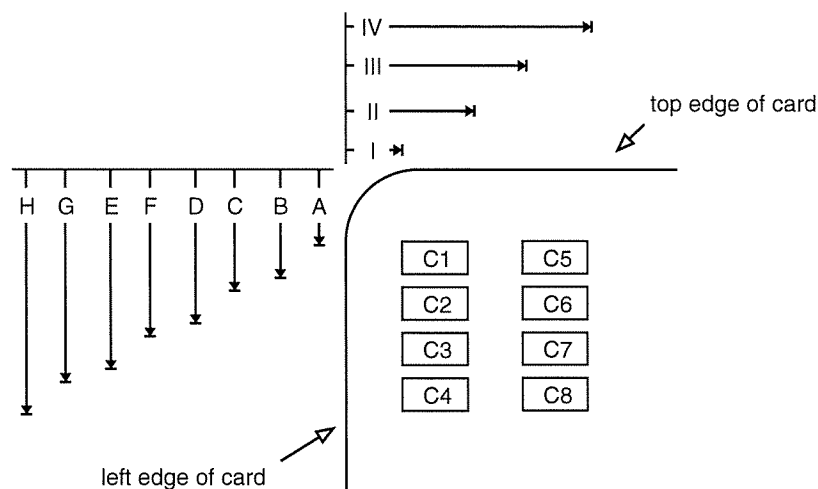


Figure 3.65 The positions of the contacts relative to the card body outline (drawing not to scale)

I	10.25 mm maximum	A	19.23 mm minimum
II	12.25 mm minimum	B	20.93 mm minimum
III	17.87 mm maximum	C	21.77 mm maximum
IV	19.87 mm minimum	D	23.47 mm minimum
		E	24.31 mm maximum
		F	26.01 mm minimum
		G	26.85 mm maximum
		H	28.55 mm minimum

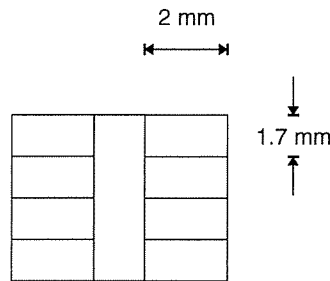


Figure 3.66 Minimum contact dimensions as specified in ISO 7816-2

connections are required. These consist of six or eight gold-plated contacts, which can be seen on every standard smart card. The locations of these contacts with respect to the card body, and their sizes, are specified by the ISO 7816-2 standard.

In France, a national standard generated by AFNOR was already in use long before ISO 7816-2 was issued. It specifies a slightly higher location for the contacts than the ISO standard. This location is also included in the ISO standard as a 'transitional contacts location', but the standard recommends that this location not be used in the future. However, since there are still many cards in France that use the 'transitional' position, it is not likely that it will disappear quickly.

The absolute position of the contact field is in the upper left corner of the card body. This is clearly shown in dimensioned drawing shown in Figure 3.65. The minimum dimensions of any contact are 1.7 mm by 2 mm (height by width). The maximum dimensions of any contact are not specified, but they are of course limited by the fact that the individual contacts must be electrically isolated from each other.

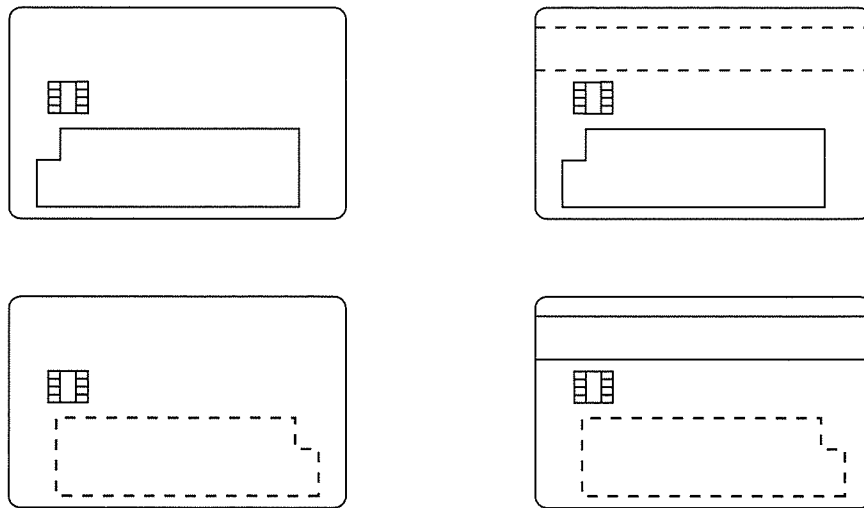


Figure 3.67 The various possible arrangements for the chip, embossing field and magnetic stripe, according to the ISO 7816-2 standard

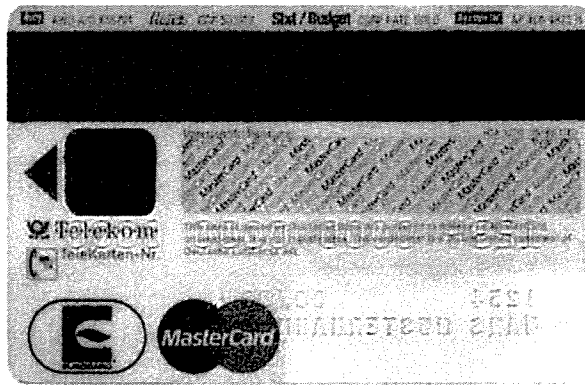


Figure 3.68 An example of a card with a chip, magnetic stripe, signature area and embossing (Source: Giesecke & Devrient)

The position of the module within the card body is specified in the standard. The locations of the magnetic stripe area and the area reserved for embossing are also exactly specified (see ISO 7811). All three of these components may be present on a single card. However, in this case the following mutual relationships must be taken into account: (a) if only a chip and an embossing field are present, they may be located on the same side or on opposite sides of the card; (b) if a magnetic stripe is also present, it and the embossing area must be located on opposite sides of the card.

3.6 CONTACTLESS CARDS

As already described in Section 2.3.3, contactless cards do not require any electrical connection between the smart card and the card terminal in order to transfer energy and data over a short distance. The most important advantages of the contactless card technology are described in Section 2.3.3. In this section we examine the technology and operating principles of contactless cards in more detail. The techniques used with contactless cards for transferring energy and data are not new. They have been common knowledge for many years in radio-frequency identification (RFID) systems, which have been used for a variety of applications, such as animal implants and transponders for electronic anti-start systems for vehicles. There are many techniques for identifying persons or objects at short or even long distances based on radio techniques, and in particular on radar techniques. Among the large variety of technical possibilities, only a small number are suitable for use in smart cards in the ID-1 format (to which we restrict our attention), since all of the functional components must be housed in a flexible card that is only 0.76 mm thick. For instance, fitting flexible batteries into the card body remains an unsolved problem for mass-produced cards. Although flexible batteries with suitable thickness are now available, there is no experience with using such batteries in the field or in mass production. Consequently, we are still limited to passive techniques in which the energy to power the card must be extracted from the electromagnetic field of the card terminal. This limits the useful range to around 1 m.

To make it easier to understand the variety of techniques used, they can be classified according to various parameters. One possibility is to classify them according to the method used to transfer energy and data. The most commonly used methods are transmission using radio waves or microwaves, optical transmission, capacitive coupling and inductive coupling. Capacitive and inductive coupling are best suited to the flat shape of a smart card lacking an internal source of power. The systems presently available on the market utilize these methods exclusively, which are also the only ones considered in the relevant group of ISO/IEC standards (10 536, 11 443 and 15 693). Consequently, in this book we limit ourselves to these methods.

Just as with contact-type smart cards, a system using contactless cards consists of at least two components, namely a card and a compatible terminal. The terminal can act as a reader or a reader/writer, according to the technology used. As a rule, the terminal includes an additional interface, via which it can communicate with a background system.

The following four functions are necessary to allow a contactless card to communicate with a terminal:

- energy transfer to the card for powering the integrated circuit
- clock signal transfer
- data transfer to the smart card
- data transfer from the smart card.

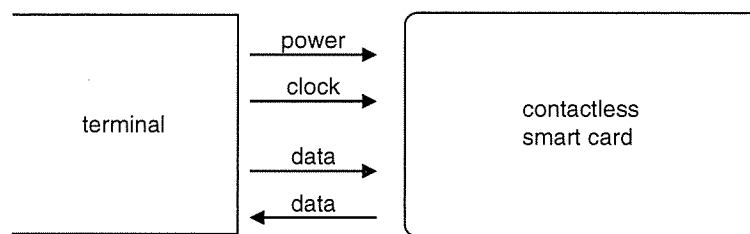


Figure 3.69 The necessary energy and data transfers between a terminal and a contactless smart card

Many different concepts based on experience with RFID systems have been developed to satisfy these requirements. Most of them are specifically designed for particular applications. For instance, there is a considerable difference between systems where the cards are only a few millimeters away from the terminal in normal use and systems where the cards can be up to a meter away from the terminal. Naturally, when many different solutions specifically designed and optimized for particular applications are developed, they are inevitably mutually incompatible.

Inductive coupling

Inductive coupling is presently the most widely used technique for contactless smart cards. It can be used to transfer both energy and data. Various requirements and constraints, such as radio licensing regulations, have resulted in a variety of actual implementations.

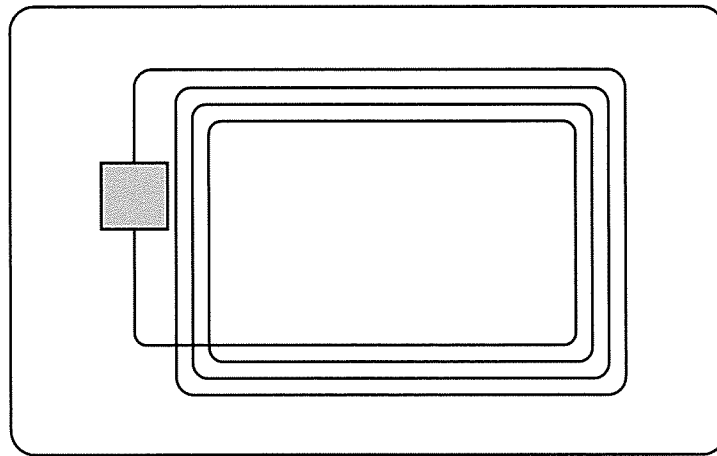


Figure 3.70 Basic construction of a contactless smart card with inductive coupling

With some applications, such as access control, it is sufficient to only be able to read the data stored in the cards, which makes technically simple solutions possible. Due to their low power consumption (a few tens of microwatts), the usable range of such cards extends to approximately one meter. Their memory capacity is usually only several hundred bits. If data must also be written, the power consumption rises to more than 100 μW . As a consequence, the range is limited to around 10 cm in the writing mode, since licensing restrictions prevent the emitted power of the writing equipment from being arbitrarily increased. The power consumption of microprocessor cards is even greater and is typically 100 mW. The distance from the terminal is thus even more restricted.

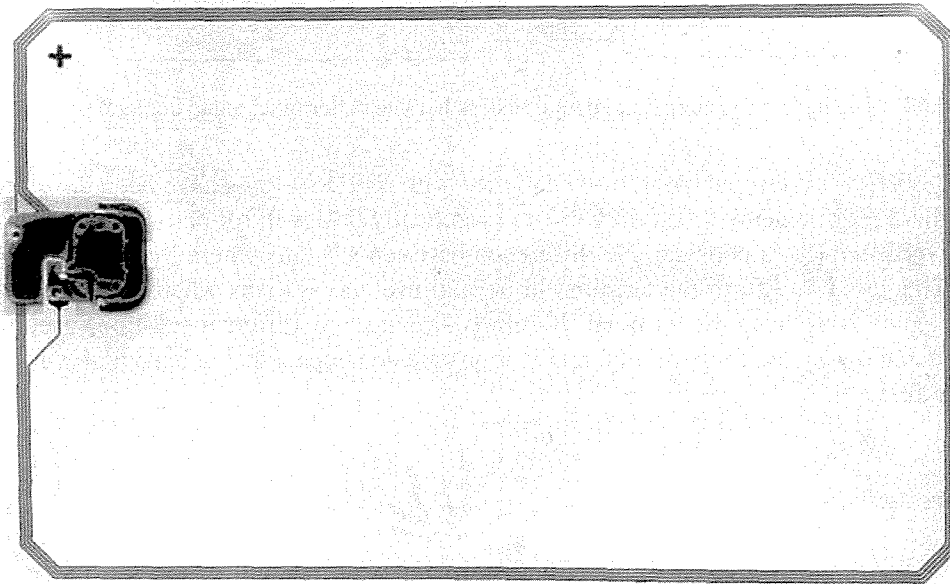


Figure 3.71 Inlay foil for a contactless smart card with inductive coupling using an etched coil

Independent of their range and power consumption, all cards that employ inductive coupling work on the same principle. One or more coils (usually with large enclosed areas) are incorporated into the card body to act as coupling components for energy and data transfers, along with one or more chips.

Energy transfer

Almost without exception, contactless smart cards are used passively. This means that all of the energy needed for operating the chip in the smart card must be transferred from the reader to the card.

This energy transfer is based on the principle of a loosely coupled transformer. A strong high-frequency magnetic field is generated by a coil in the terminal in order to transfer the energy. The most commonly used frequencies are <135 kHz and 13.56 MHz, which correspond to wavelengths of 2400 m and 22 m, respectively. The wavelengths of the electromagnetic fields are thus several times greater than the distance from the card to the terminal, which means that the card is located in the near field of the terminal. This allows the loosely coupled transformer model to be used. If a contactless card is brought close to the terminal, a portion of the terminal's magnetic field passes through the coil in the card and induces a voltage U_i in this coil. This voltage is rectified to provide power to the chip. Since the coupling between the coils in the terminal and the card is very weak, the efficiency of this arrangement is very low. A high current level is thus required in the terminal coil to achieve the necessary field strength. This is achieved by connecting a capacitor C_T in parallel with the coil L_T , with the value of the capacitor chosen such that the coil and capacitor form a parallel-resonant network whose resonant frequency matches the frequency of the transfer signal.

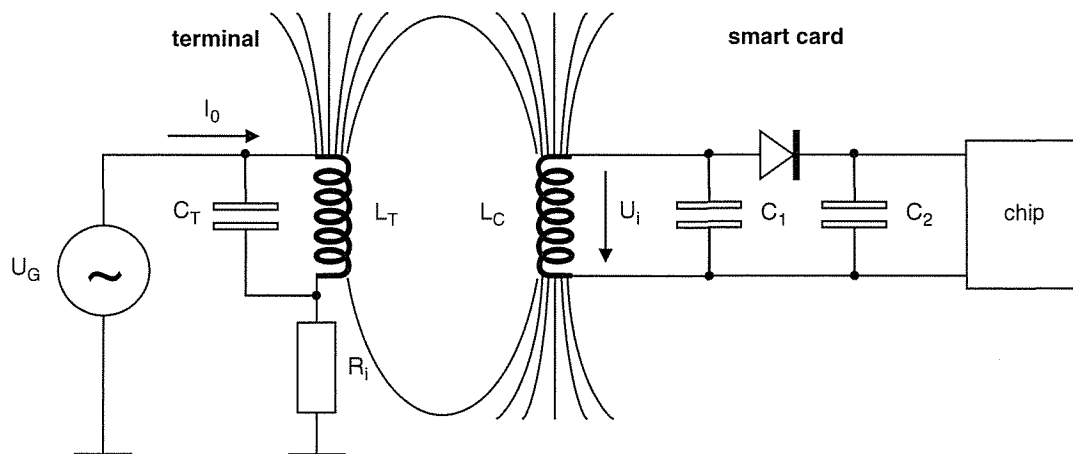


Figure 3.72 Using inductive coupling to supply energy to a smart card

Coil L_C and capacitor C_1 in the card also form a resonant circuit with the same resonant frequency. The voltage induced in the card is proportional to the signal frequency, the number of windings of coil L_C and the enclosed area of the coil. This means that the number of turns needed for the coil drops with increasing signal frequency. At 125 kHz, it is 100 to 1000 turns, while at 13.56 MHz it is only 3 to 10.

Data transfer

For transferring data from the terminal to the card, all known digital modulation techniques can be used. The most commonly used techniques are:

- ASK (amplitude-shift keying)
- FSK (frequency-shift keying)
- PSK (phase-shift keying).

ASK and PSK are usually used, since these are especially easy to demodulate.

In the other direction, from the smart card to the terminal, a type of amplitude modulation is used. It is generated by using the data signal to digitally alter a load in the card (load modulation). If a smart card tuned to the resonant frequency of the terminal is brought into the near field of the terminal, it draws energy from this field as previously described. This causes the current I_0 in the coupling coil of the terminal to increase, which can be detected as an increased voltage drop across an internal resistor R_i . The smart card can thus vary (amplitude modulate) the voltage U_0 in the terminal by varying the load on its coil, for example by switching the load resistor R_2 into and out of the circuit as shown in Figure 3.73. If the switching of resistor R_2 is controlled by the data signal, the data can be detected and evaluated in the terminal.

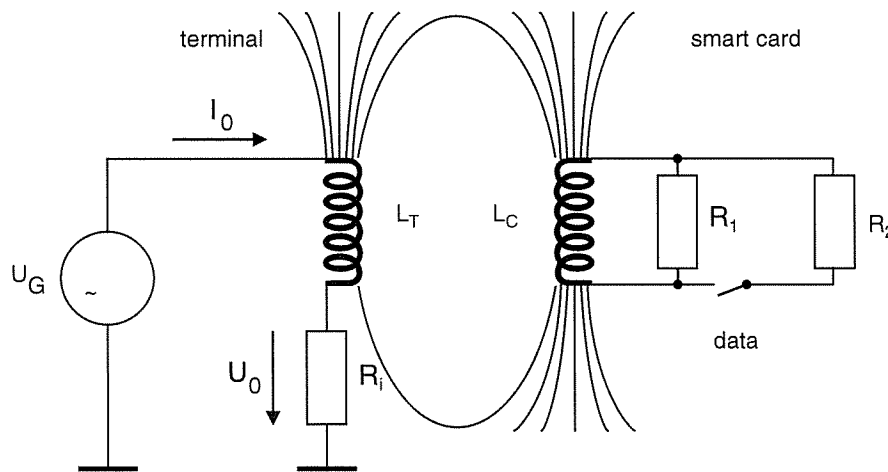


Figure 3.73 A sample circuit illustrating the principle of load modulation, which is used in contactless smart cards for data transmission

Due to the low degree of coupling between the coils in the terminal and the card, the voltage variations induced in the terminal by load modulation are very small. In practice, the amplitude of the usable signal is only a few millivolts. This can only be detected using sophisticated circuitry, since it is overlaid by the significantly larger signal (around 80 dB) transmitted by the terminal. However, if a subcarrier frequency is employed with a frequency of f_s , the received data signal appears in the terminal as two sidebands at the frequencies $f_c \pm f_s$. These can be isolated from the significantly stronger terminal signal by filtering with a bandpass filter and then amplified. After this, they can readily be demodulated. The

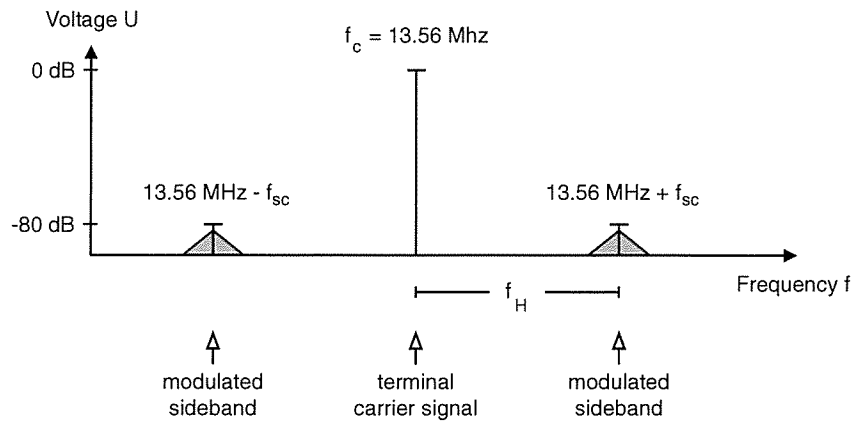


Figure 3.74 Load modulation using a subcarrier produces two sidebands separated from the transmission frequency of the terminal by the value of the subcarrier frequency f_s . The information is contained in the sidebands of the two subcarrier sidebands, which are produced by modulation of the subcarrier (based on Klaus Finkenzeller [Finkenzeller 02])

disadvantage of modulation with a subcarrier is that it requires significantly more bandwidth than direct modulation. It can thus only be used in a limited number of frequency bands.

Capacitive coupling

If the distance between the card and the terminal is very small, it is possible to transfer data using capacitive coupling. With this type of coupling, conductive surfaces are incorporated

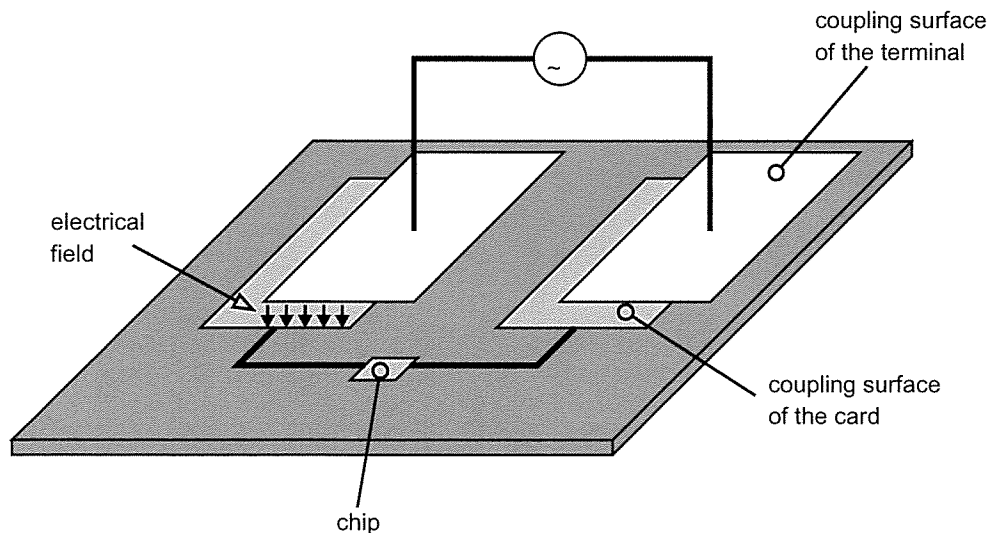


Figure 3.75 Operating principle of capacitive coupling. The coupling arises from the alternating electrical field between two parallel, electrically conductive surfaces in the card and terminal

into the card body and the terminal such that they act as the plates of a capacitor when the card is inserted in the terminal or placed on the terminal. The capacitance that can be obtained essentially depends on the sizes of the coupling surfaces and their separation. The maximum size is thus limited by the dimensions of the card, while the minimum separation is determined by the insulation required between the coupling surfaces. With an acceptable level of cost and effort, a usable capacitance of several tens of picofarads can be obtained. This is insufficient for transferring enough energy to power a microprocessor. Consequently, this method is used only for data transmission, with the operating power being transferred inductively. This mixed method has been standardized in ISO/IEC 10 536 for 'close coupling cards', and it is fully described in Section 3.6.1. As its name says, this method is limited to small coupling distances.

Collision avoidance

When contactless cards are used, there is always a possibility that two or more cards may be located in the range of a terminal at the same time. This is especially true for systems with large effective ranges, but it can even happen with systems with relatively small ranges – for instance, two cards might be lying on top of each other and thus be activated concurrently by the terminal. All cards within range of a particular terminal will attempt to respond to commands from the terminal. However, simultaneous data transmissions will unavoidably cause interference and loss of data if suitable countermeasures are not taken. The technical methods used to ensure interference-free data exchanges with multiple cards within the effective range of a card terminal are called collision-avoidance methods or anticollision methods.

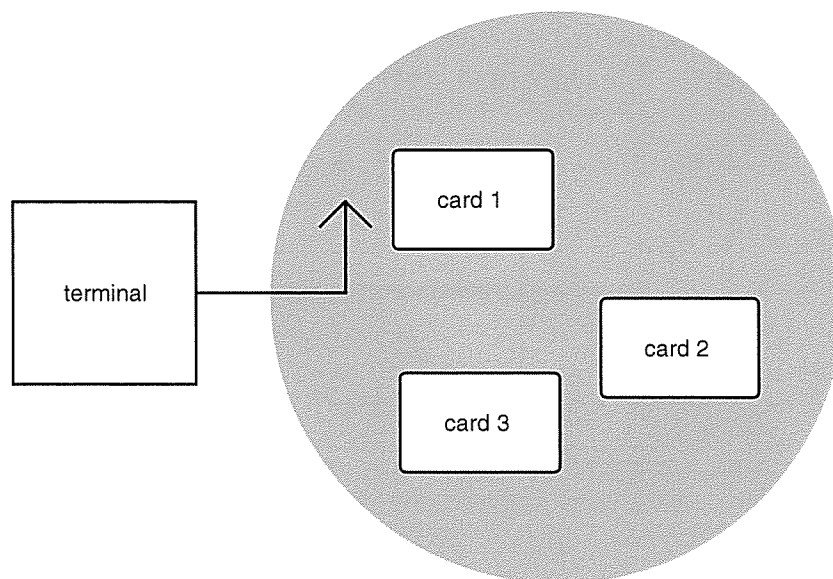


Figure 3.76 Concurrent operation of several cards within the range of a terminal (multiple access) requires using an anticollision method to ensure interference-free data exchanges

Exchanging data between many mobile units and a base station is a frequently encountered situation in communications engineering, and it is referred to as 'multiple access'. A typical

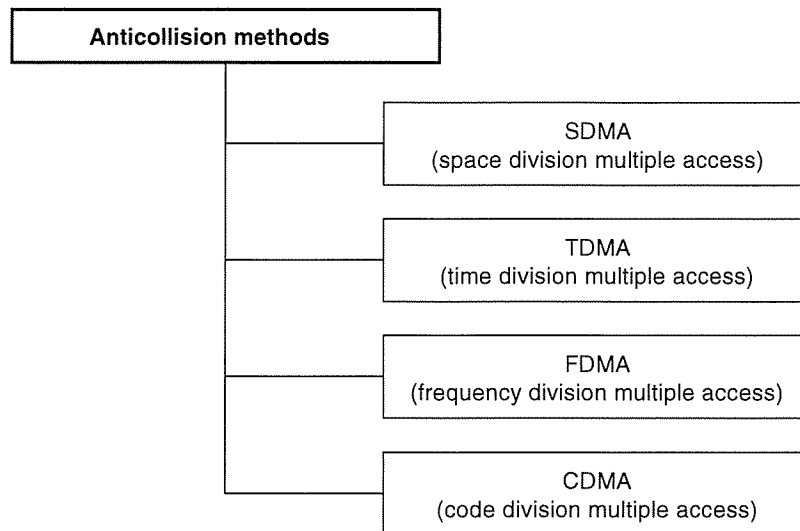


Figure 3.77 The four types of anticollision methods

example is a mobile telephone network, in which all users located in a particular radio cell concurrently access a single base station. Numerous methods have been developed to allow the signals of the individual users to be distinguished from each other. These anticollision methods can be classified into four types, as shown in Figure 3.77.

Space division multiple access (SDMA) attempts to limit or scan the operational area of a terminal in such a way that only one card can be acquired at any given time. Since this method requires very complicated and correspondingly expensive aerials, it is not used for contactless cards.

With time division multiple access (TDMA), measures are taken to ensure that the individual cards have different timing behavior so that they can be separately identified and individually addressed by the terminal. This is the most commonly used method, and it has many variants. Two of them, which are standardized in ISO/IEC 14 433-3 for 'proximity cards', are described extensively in Section 3.6.3.

With frequency division multiple access (FDMA), different carrier frequencies are provided concurrently for multiple transmission channels. However, this technique is technically complicated and thus expensive. Consequently, it is not used for contactless cards. The same considerations also apply to code division multiple access (CDMA).

The present state of standardization

Given the many different techniques used by various manufacturers, standardization (which was initiated in 1988 by ISO/IEC) proved to be difficult and time consuming, as was expected. The responsible working group had the task of defining a standard for contactless cards that is largely compatible with other standards for identification cards. This means that a contactless card can also have other functional components, such as a magnetic stripe, embossing and chip contacts. This allows contactless cards to also be used in existing systems that employ other technologies. As already described, the technical options for transferring energy and

data without using contacts essentially depend on the desired distance between the card and the terminal for reading and writing data. It was therefore not possible to create a single standard that provides a single technical solution to all the requirements arising from various applications.

Presently, three different standards describing three different reading ranges have been completed. Each of these standards in turn permits various technical solutions, since the members of the standardization committee could not agree on a single solution. In order to achieve interoperability among the various options, card terminals must support all of these options.

Table 3.6 Completed ISO/IEC standards for contactless smart cards

Standard	Type of contactless smart card	Range
ISO/IEC 10 536	Close-coupling card	Up to approx. 1 cm
ISO/IEC 14 443	Proximity coupling card (PICC)	Up to approx. 10 cm
ISO/IEC 15 693	Vicinity coupling card (VICC)	Up to approx. 1 m

Standardization started with ‘close-coupling’ cards (ISO/IEC 10536), since the microprocessors available at that time had relatively high power consumption, making energy transfer over a relatively large distance impossible. The essential parts of this standard have been completed and approved and are described in the following section. In use, this type of card offers only minimal advantages compared with normal contact-type cards, since it must be inserted into a terminal or at least precisely placed on a surface of a card terminal. Furthermore, the structure of the card is complex, which results in high manufacturing costs. Consequently, up to now this type of system has hardly established a significant position in the market.

3.6.1 Close-coupling cards: ISO/IEC 10536

In the ISO/IEC 10536 standard for close-coupling cards, this application is designated as ‘slot or surface operation’, which expresses the fact that in use the card must be inserted into a slot or laid on a marked surface of the terminal. The ISO/IEC 10536 standard, which bears the title ‘Identification Cards – Contactless Integrated Circuit(s) Cards’, consists of four parts:

- Part 1: Physical characteristics
- Part 2: Dimension and location of coupling areas
- Part 3: Electronic signals and reset procedures
- Part 4: Answer to reset and transmission protocols.

Parts 1 through 3 have already become international standards, while Part 4 is still in preparation. The important ingoing requirements for these standards were the following:

- extensive compatibility with ISO 7816
- operation with arbitrary orientation of the card to the reader

- transfer carrier frequency between 3 and 5 MHz
- bidirectional data transmission with inductive or capacitive coupling
- card power consumption less than 150 mW (adequate for microprocessor chips).

Part 1 of the standard defines the physical characteristics of the card. Essentially the same requirements are imposed as for contact-type smart cards, particularly with regard to bending and twisting. One difference is in the tolerance to electrostatic discharge. Since a contactless card does not require any conductive path between the card surface and the integrated circuit embedded in the card body, it is largely insensitive to damage from ESD. A test voltage of 10 kV is thus specified in the standard, compared with 1.5 kV for cards with contacts.

Part 2 of the standard specifies the locations and dimensions of the coupling components. Since it was not possible to agree on a single method, both capacitive and inductive coupling

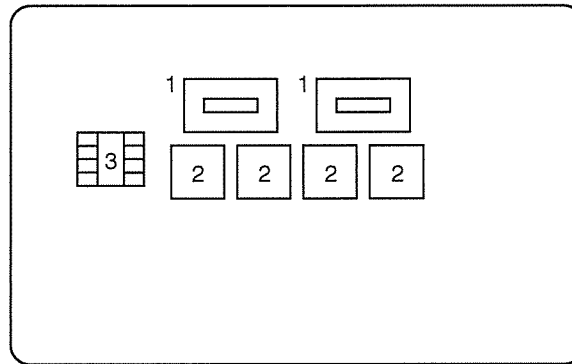


Figure 3.78 The arrangement of the coupling components of a contactless smart card: (1) coupling coils in the card body, (2) capacitive coupling surfaces in the card body and (3) a set of contacts for the chip

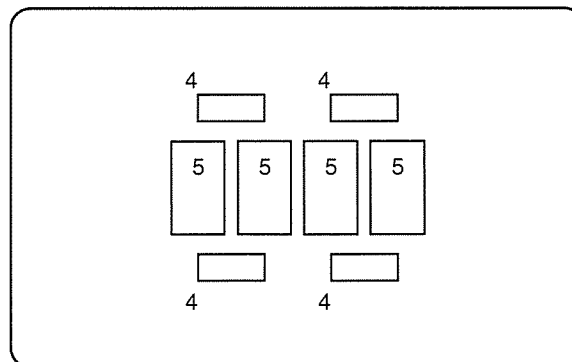


Figure 3.79 The arrangement of the coupling components in a terminal for contactless smart cards: (4) coupling coils in the terminal, (5) capacitive coupling surfaces in the terminal

components are defined in such a way that both can be implemented together in a single card or terminal. Examples of this are shown in Figures 3.78 and 3.79. The chosen arrangement is intended to ensure orientation independence with suitable excitation in the terminal.

Part 3 of the standard, published in 1996, is the most important part to date. It describes the modulation methods to be used for capacitive and inductive data transmission, since agreement on a single method could not be achieved. A terminal that complies with the standard must therefore support both methods, and both methods may be implemented in a single card.

Energy transfer

Energy is transferred by a sinusoidal alternating magnetic field with a frequency of 4.9152 MHz, which passes through one or more inductive coupling surfaces, depending on how many coupling coils are present in the card. The terminal must generate all four fields.

Alternating magnetic fields $F1$ and $F2$, which pass through areas $H1$ and $H2$, have a mutual phase difference of 180 degrees, as do fields $F3$ and $F4$, which pass through areas $H3$ and $H4$. The phase difference between fields $F1$ and $F3$ and between $F2$ and $F4$ is 90 degrees. Each magnetic field is strong enough to transfer at least 150 mW to the card. However, the card should not consume more than 200 mW. This complicated definition of the magnetic fields is necessary to achieve the same data transfer characteristics for four different card orientations, as explained below.

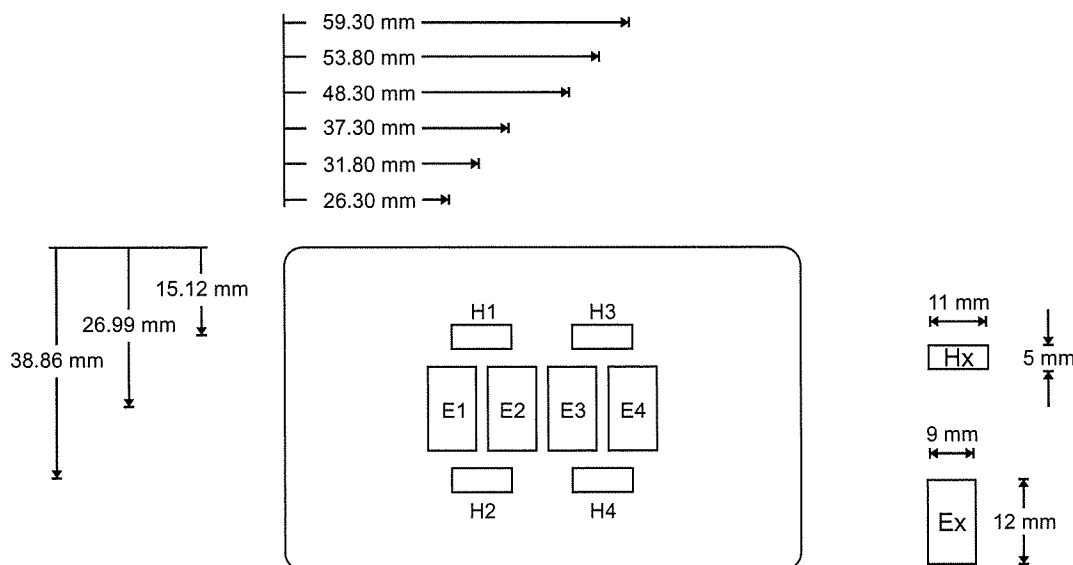


Figure 3.80 Locations and sizes of the coupling areas in the contactless card and the terminal

3.6.1.1 Inductive data transfer

Different types of modulation are used for data transmission in the two directions.

Data transmission from the card to the terminal

For data transmission from the card to the terminal, a 307.2 kHz subcarrier is first generated using load modulation (see Figure 3.81), with a load variation of at least 10 %. Data modulation is achieved by switching the phase of the subcarrier by 180 degrees, producing two phase states that can be interpreted as logic 1 and logic 0. The initial state after the magnetic field has been established is defined to be logic 1. This initial state (interval t_3 in Figure 3.84) remains stable for at least 2 ms. Following this, every subcarrier phase shift represents a reversal of the logic state, yielding non-return to zero (NRZ) coding. The data transmission rate, at least for the ATR, is 9600 bits per second.

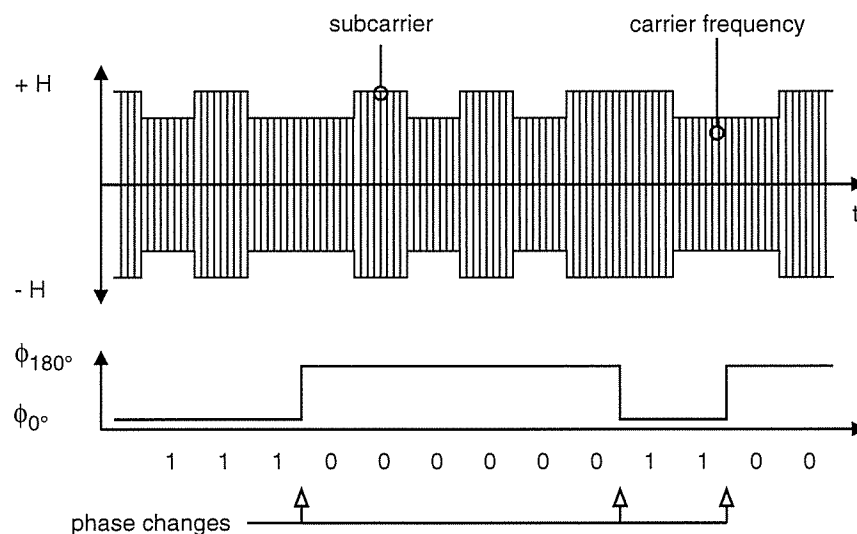


Figure 3.81 Operating principle of phase modulation for data transmission with a contactless smart card. The upper diagram shows the alternating magnetic field, and the associated phase states are shown in the lower diagram. The carrier frequency is 4.9152 MHz, and the subcarrier frequency is 307.2 kHz

Data transmission from the terminal to the card

To transfer data from the terminal to the card, the four alternating magnetic fields $F1$ through $F4$, which pass through coupling surfaces $H1$ through $H4$, are phase modulated using phase-shift keying (PSK). This causes the phase of all four fields to simultaneously shift by 90 degrees. In this way, two phase states A and A' are defined. Depending on the orientation of the card relative to the terminal, this yields two different constellations of phase states, as shown in Figures 3.82 and 3.83.

Since the card must work in all four possible orientations with respect to the terminal, the initial state (intervals t_2 and t_3 in Figure 3.84) is interpreted as a logic 1, regardless of which

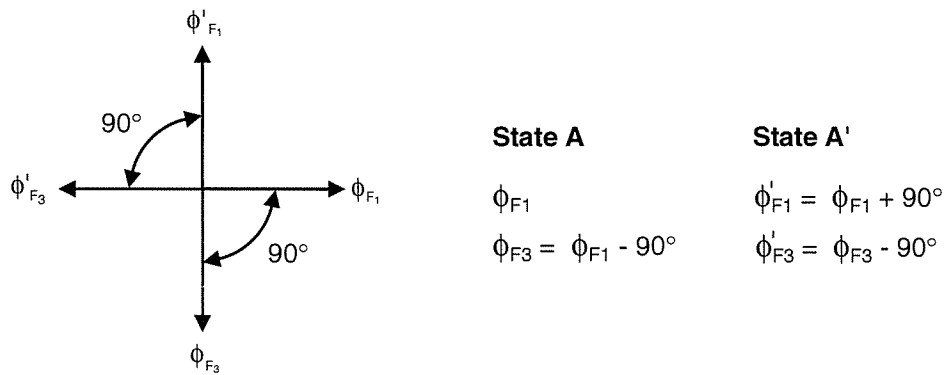


Figure 3.82 The first phase modulation variant for data transmission with a contactless chip card. The four arrows represent phase vectors

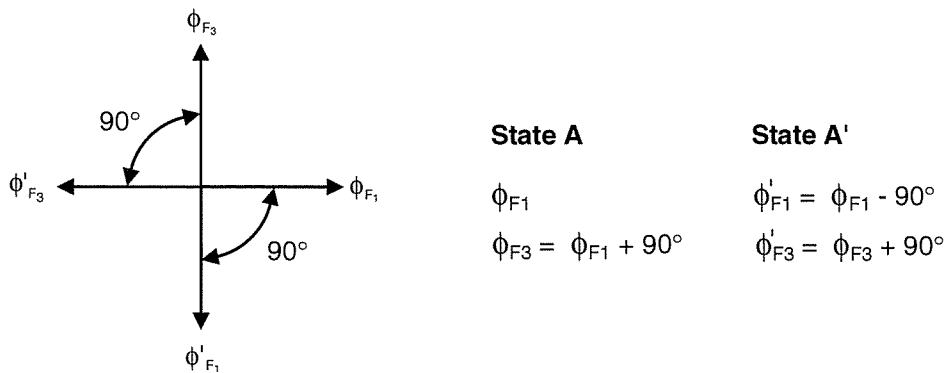


Figure 3.83 The second phase modulation variant for data transmission with a contactless chip card. The four arrows represent phase vectors

of the indicated alternatives is actually present. Following this, every phase change represents a reversal of the logic state, which again produces an NRZ encoding.

3.6.1.2 Capacitive data transfer

For capacitive data transmission from the card to the terminal, one pair of coupling surfaces is used, depending on the orientation of the card relative to the terminal – either *E1* and *E2* or *E3* and *E4* as shown in Figure 3.80. The other pair of coupling surfaces can be used for data transmission in the opposite direction. Since the card sends the ATR via one particular pair of coupling surfaces, the terminal can recognize the relative orientation of the card. The maximum potential difference between a pair of coupling surfaces is limited to 10 V, but it must at least exceed the minimum differential voltage of the receiver (± 300 mV). Differential NRZ encoding is used for data transmission. The transmitter generates the encoding by reversing the voltage between surfaces *E1* and *E2* or *E3* and *E4*. The state representing a logic 1 is again established in interval t_3 (see Figure 3.84). Following this, every polarity reversal represents a change in the logic state.

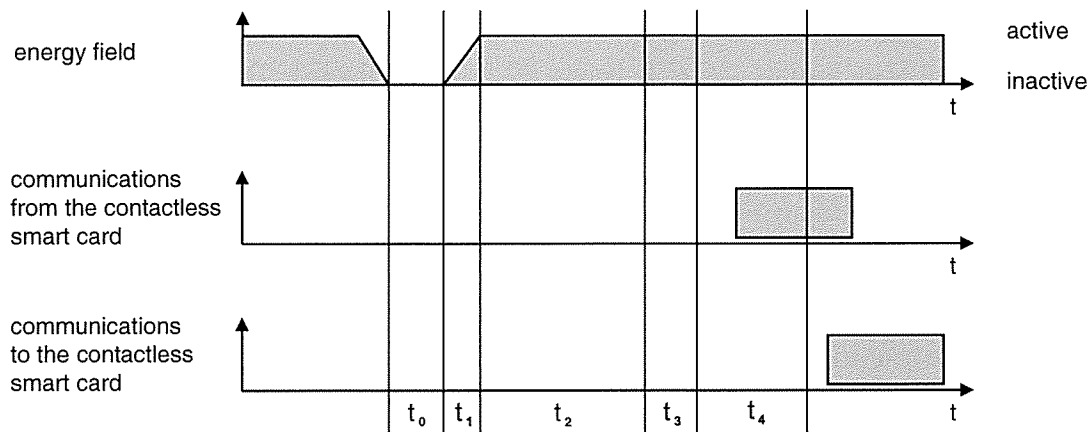


Figure 3.84 Timing diagram for data transmission with a contactless smart card according to ISO/IEC 10536 3. Here $t_0 \geq 8$ ms, $t_1 \leq 0.2$ ms, $t_2 = 8$ ms, $t_3 = 2$ ms and $t_4 \leq 30$ ms

Initial state and answer to reset (ATR)

In order for the terminal to unambiguously determine the type of data transmission and the orientation of the card at the beginning of a data exchange, certain time intervals must be defined for initiating energy and data transfers. Figure 3.84 shows the constraints and values for the reset recovery time t_0 , power-up time t_1 , initialization time t_2 , stable logic state time t_3 and answer to reset time t_4 .

Minimum reset recovery time: t_0

If a reset is to be produced by switching the energy-transfer field off and back on, the time between switching the field off and then on again, during which no energy is transferred, must be equal to or greater than 8 ms.

Maximum power-up time: t_1

The time required for the energy-transfer field produced by the terminal to be established must be less than or equal to 0.2 ms.

Initialization time: t_2

The initialization time, which is the time allowed for the card to attain a stable operating state, is 8 ms.

Stable logic state time: t_3

Prior to the Answer to Reset, the logic state is held at the logic 1 level for 2 ms. During this interval, the card and the terminal are set to logic 1 for inductive data transmission.

Maximum response time for ATR: t_4

The card must start sending the ATR before 30 ms have elapsed. The card can use the ATR to indicate that the conditions for subsequent operation must be changed with regard to energy level, data transmission rate or the frequency of the fields. The ‘maneuvering room’ provided here can be utilized according to the requirements of the application. For instance, a significantly higher data transmission rate can be selected for a time-critical application.

3.6.2 Remote-coupling cards

The term ‘cards with remote coupling’ encompasses smart cards that can transmit data over a distance of a few centimeters to approximately one meter from the terminal. This capability is of great significance for all applications in which data should be exchanged between the card and the terminal without requiring the card user to take the card in his or her hand and insert it into a terminal. Some sample applications are:

- access control
- vehicle identification
- electronic driver’s licenses
- ski passes
- airline tickets
- electronic purses
- baggage identification.

The variety of applications suggests that there are a large number of possible technical implementations. In the preparation of the standards, an attempt was made to limit the number

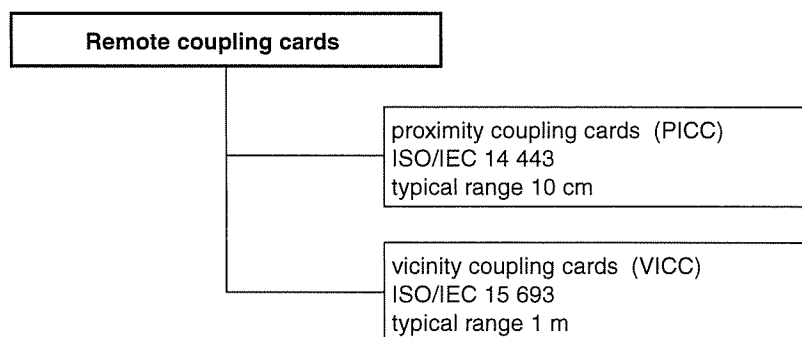


Figure 3.85 For remote-coupling cards, the ISO/IEC standards distinguish between proximity cards and vicinity cards

of technical variants, with only mixed success. International standards ISO/IEC 14 443 and ISO/IEC 15 633 cover the ranges of up to 10 cm and 1 m, respectively.

3.6.3 Proximity integrated circuit(s) cards: ISO/IEC 14 443

The ISO/IEC 14 443 standard, which is titled 'Identification cards – Contactless integrated circuit(s) cards – Proximity cards', describes the properties and operation modes of contactless smart cards with a range of approximately 10 cm. The amount of energy that can be transferred over this range is sufficient to power a microprocessor. In order to allow this type of card to be used with existing infrastructures for contact-type cards, they often have contacts in addition to the coupling components for contactless operation, so that they can be used with or without contacts as desired. This type of card is called a 'dual-interface card' or 'combicard'.

The ISO/IEC 14 443 standard consists of the following parts:

- Part 1: Physical characteristics
- Part 2: Radio frequency power and signal interface
- Part 3: Initialization and anticollision
- Part 4: Transmission protocol.

Physical characteristics

The physical characteristics of proximity cards, which are defined in Part 1 of the ISO/IEC standard for proximity integrated circuit cards (PICCs), essentially correspond to the requirements specified for smart cards with contacts. It is to be expected that in use, proximity cards will be exposed to electromagnetic fields corresponding to those intended to be used for the operation of other types of cards that comply with standards such as ISO/IEC 10 536 or ISO/IEC 15 693. The cards must not suffer permanent damage as the result of exposure to such fields or the environmental stress of normal ambient electromagnetic fields. In order to ensure this, the standard specifies maximum values for stresses due to alternating electric and magnetic fields that the cards must withstand without damage. It is the task of the semiconductor manufacturer to design the chips such that they meet these requirements.

Radio-frequency power and signal interface

Proximity cards work on the principle of inductive coupling. Operating power and data are both transferred using an alternating magnetic field generated by the card terminal. In the ISO/IEC 14 443 standard, the card terminal is called a 'proximity coupling device' (PCD). For the sake of readability, in the following description the more general term 'terminal' and 'PCD' are used interchangeably.

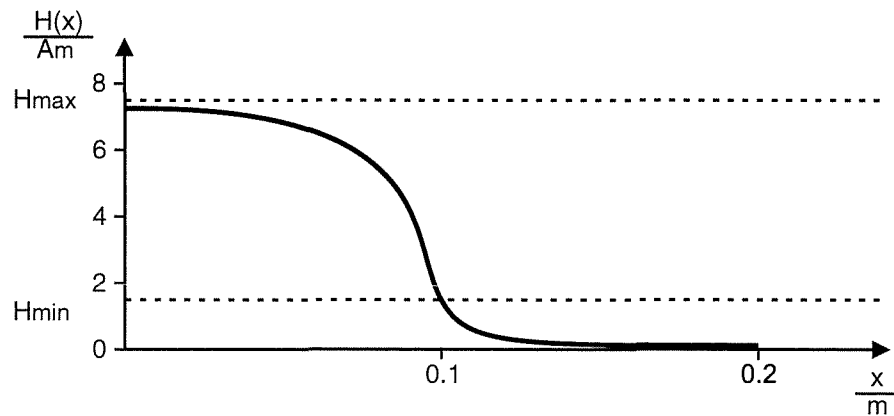


Figure 3.86 Typical magnetic field strength characteristic of a terminal for proximity cards (PCD)

The transmission frequency of the PCD is set to $f_C = 13.56 \text{ MHz} \pm 7 \text{ kHz}$, with a magnetic field strength H of at least 1.5 A/m and at most 7.5 A/m (effective value). The typical field strength versus distance is shown in Figure 3.86.

The range of the system can be estimated from the field strength of the PCD and the activation field strength of a proximity card (PICC). With the typical field strength curve shown in Figure 3.86 and an assumed PICC activation field strength of 1.5 W/m , we obtain a range of approximately 10 cm .

Signal and communication interface

Two different communication interfaces are defined in the ISO/IEC 14 443 standard, which are designated Type A and Type B. The reason for standardizing two different methods was not technical, but rather that at the time that ISO/IEC 14 443 was being prepared, various designs from different manufacturers were already in existence. As is often the case with standardization, the differing interests of the persons involved made it impossible for them to agree on a single method, although that would have been technically desirable. The two methods mentioned above were agreed on as a compromise, and they were published as an international standard in June 2001. Even with the already existing methods, it is necessary for card terminals to support both methods in order to achieve full interoperability with all cards meeting the ISO/IEC 14 433 standard, since the cards generally support only one of the two standard methods.

While a terminal is waiting to detect a proximity card, it must periodically switch back and forth between the two communications methods. This allows it to recognize both Type-A and Type-B cards. Once the PCD has recognized a card, it continues to use the appropriate communications method until the card is deactivated by the terminal or leaves the working range of the terminal.

carrier amplitude envelope

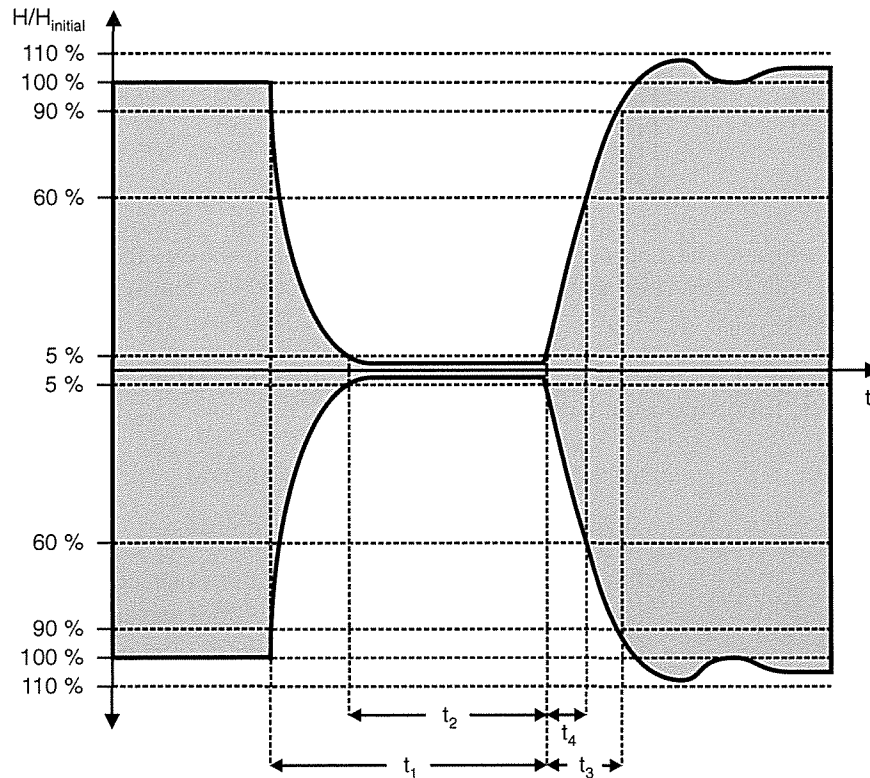


Figure 3.87 Specification of the blanking interval (gap) according to ISO/IEC 14433-2. The maximum duration of the gap is limited to 3 μs in order to interrupt the energy supply to the card as briefly as possible. Here $2.0 \mu\text{s} \leq t_1 \leq 3.0 \mu\text{s}$; $0.5 \mu\text{s} \leq t_2 \leq t_1$ if $t_1 > 2.5 \mu\text{s}$, or $0.7 \mu\text{s} \leq t_2 \leq t_1$ if $t_1 \leq 2.5 \mu\text{s}$; and $0 \mu\text{s} \leq t_4 \leq 0.4 \mu\text{s}$

3.6.3.1 Type-A communications interface

With Type-A cards, data transmission takes place in both directions at a bit rate of $f_C/128$ ($\approx 106 \text{ kbit/s}$).

Data transmission from the terminal to the card

Digital amplitude modulation (100% ASK) with modified Miller coding is used for data transmission from the PCD to the card, with the length of the blanking interval (gap) being limited to 3 μs . This relatively short blanking interval makes it easier to provide a steady supply of energy to the card. The exact specification of the length of the blanking interval and its rise and decay characteristics are shown in Figure 3.87.

The card recognizes the end of the pause during interval t_4 , which means after the magnetic field has reached 5% of H_{INITIAL} and before it exceeds 60% of H_{INITIAL} . Overshoots must be limited to $H_{\text{INITIAL}} \pm 10\%$.

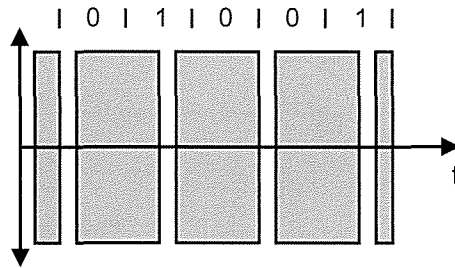


Figure 3.88 Coding of a bit sequence transmitted from the terminal to the card for a Type-A communications interface with 100 % ASK and modified Miller coding at 106 kbit/s. The figure shows the voltage at the terminal aerial

An example of the coding of a bit sequence using modified Miller coding is shown in Figure 3.88. The following coding rules apply here:

- logic 1: blanking interval after half the bit interval
- logic 0: no blanking, with the following exceptions:
 - if there are two or more logic 0 states in succession, there is a blanking interval at the start of the bit interval
 - if the first bit of a protocol frame is a 0, it is represented by a blanking interval at the start of the bit interval
- start of a message: blanking interval at the start of a bit interval
- end of a message: logic 0 followed by one bit with no blanking interval
- no data: no blanking interval for the duration of at least two bits.

Data transmission from the card to the terminal

The bit rate for data transmission from the card to the terminal is also $f_C/128$ (≈ 106 kbit/s). Load modulation with a subcarrier is used, which means that the subcarrier is generated by switching a load inside the card. The frequency of the subcarrier is specified to be $f_S = f_C/16$ (≈ 847 kHz). The subcarrier is modulated by switching the subcarrier on and off (on-off keying, or OOK) using Manchester coding. An example of the coding of a bit sequence is shown in Figure 3.89.

The coding is defined as follows:

- logic 1: the carrier is modulated by the subcarrier during the first half of the bit interval
- logic 0: the carrier is modulated by the subcarrier during the second half of the bit interval
- start of a message: the carrier is modulated by the subcarrier during the first half of the bit interval
- end of a message: the carrier is not modulated for one-bit interval
- no data: no subcarrier modulation.

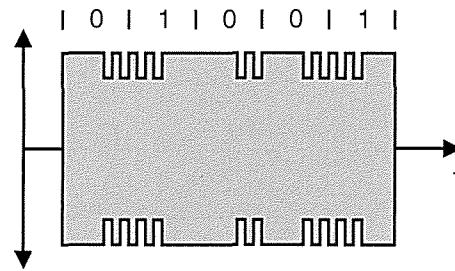


Figure 3.89 Load modulation for data transmission from the card to the terminal using a subcarrier at a frequency of $f_C/16$ (≈ 847 kHz) and Manchester coding with a bit rate of 106 kbit/s and OOK. The figure shows the voltage on the card coil

3.6.3.2 Type-B communications interface

Data transmission from the terminal to the card

With Type-B cards, ASK modulation with a modulation index of 10 % (-2 %, $+4$ %) is used for data transmission from the PCD to the card. In contrast to the Type-A method, in which continuity of the energy supply is assured by very short blanking intervals, with the Type-B method it is assured by the small modulation index, which is defined such that at least 86 % of the carrier field is always available. The bit rate is again $f_C/128$ (≈ 106 kbit/s). The exact form of Type-B modulation is shown in Figure 3.90.

carrier amplitude envelope

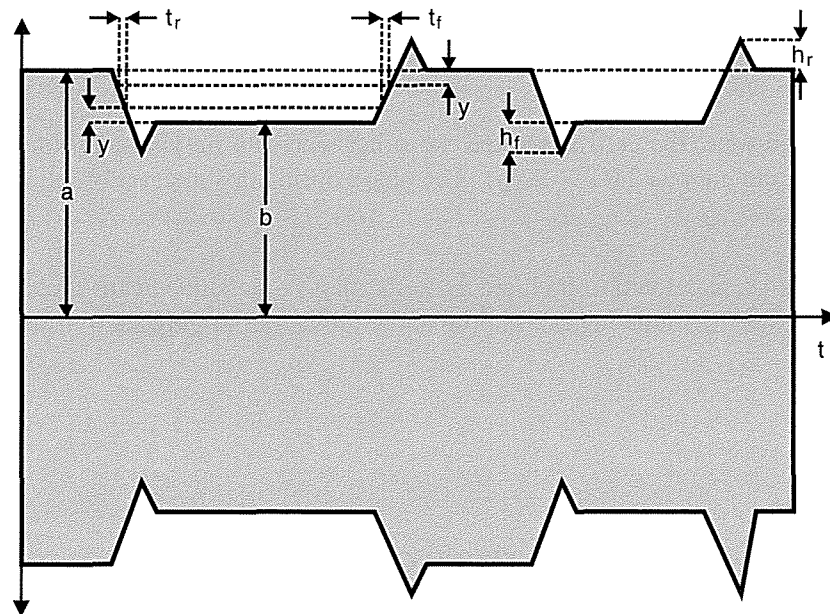


Figure 3.90 Type-B carrier modulation. A continuous supply of energy is made possible by the small modulation index (10 %). Here $t_r, t_f \leq 2 \mu\text{s}$, $y = 0.1 \cdot (a - b)$ and $h_f, h_r \leq 0.1 \cdot (a - b)$

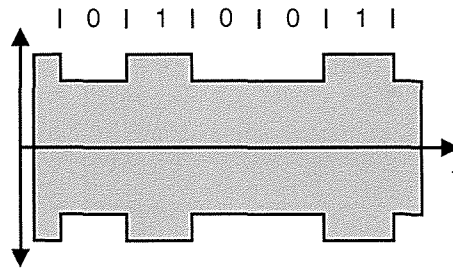


Figure 3.91 Coding of a bit sequence from the terminal to the card for a Type-B communications interface with 10 % ASK, NRZ coding and a bit rate of 106 kbit/s. The figure shows the voltage on the terminal aerial

Simple non-return to zero (NRZ) bit coding is used, with the following coding rules:

- logic 1: high carrier amplitude
- logic 0: low carrier amplitude.

Data transmission from the card to the terminal

With the Type-B method, load modulation with a subcarrier is also used for data transmission from the card to the terminal. The frequency of the subcarrier is again $f_C/16$ (≈ 847 kHz). In contrast to Type A, the subcarrier is modulated by shifting the phase by 180 degrees (binary phase-shift keying, or BPSK), again using a bit rate of $f_C/128$ (≈ 106 kbit/s) and NRZ coding. In order to have an unambiguous initial state, the following sequence must be observed at the start of each protocol frame:

- No subcarrier is generated during a guard time interval $T_{RO} > 64/f_S$ following reception of data from the terminal.
- After the guard time, the card generated the subcarrier with no phase shifting for a synchronization time interval $T_{R1} > 80/f_S$. The phase during this interval is defined to be the reference phase ϕ_0 .

The initial phase ϕ_0 is defined to be logic 1, so the first phase shift means a change from a logic 1 to a logic 0. The following rules apply to the remainder of the data transmission:

- logic 1: ϕ_0
- logic 0: $\phi_0 + 180^\circ$

3.6.3.3 Initialization and anticollision (ISO/IEC 14 433-3)

When a proximity card comes within the working range of a terminal, communications between the card and the terminal must first be established. It may happen that the terminal is already communicating with another card, or that several cards are concurrently present

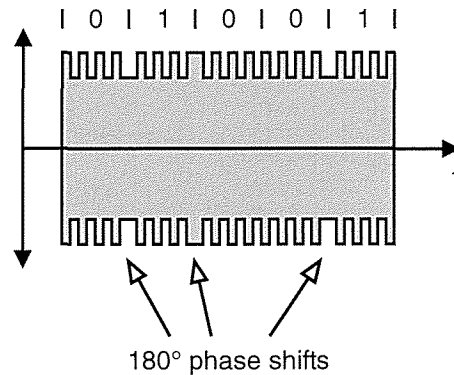


Figure 3.92 Coding of a bit sequence from the card to the terminal for a Type-B communications interface with a subcarrier of $f_c/16$ (≈ 847 kHz), BPSK, NRZ coding and a bit rate of 106 kbit/s. The figure shows the voltage on the coil in the card

within the working range of the terminal. Means must be provided to allow interference-free communication with a single card or a specific group of cards to occur under such conditions.

Establishing communications between a card and a terminal and the anticollision methods to be used for selecting a individual card are described in Part 3 of ISO/IEC 14 433. Due to the use of different modulation methods, Type-A and Type-B cards also have different protocol frames and anticollision methods.

Type-A initialization and anticollision

A dynamic binary search algorithm is used to initialize and select Type-A cards. With this method, it is necessary for the terminal to be able to recognize a data collision at the bit level. As explained below, the Manchester coding used here makes bitwise collision detection possible (see Figure 3.97). However, this requires all cards within the working range of the terminal to transmit their data synchronously.

If a proximity card comes into the field of a terminal, the microprocessor in the card is supplied with power, and following the power-on reset the card enters the Idle state. In this state, the card is only allowed to respond to a REQA (Request Type-A) command or a WUPA (Wake-up Type-A) command. All other commands transmitted by the terminal for communicating with any other Type-A or Type-B cards already present within the working range of the terminal must be ignored in order to avoid interfering with these communications. The state diagram shown in Figure 3.93 shows all possible states that can be assumed by a Type-A card during the initialization and anticollision phase.

As already mentioned, the card enters the Idle state after the power-on reset. The standard requires the card to enter the Idle state within 5 ms after it receives adequate operating power from the terminal's field. In the Idle state, the card awaits further commands. It changes to the Ready state when it recognizes a REQA or WUPA command, but it ignores all other commands.

In order to ensure a high level of reliability for recognizing the REQA and WUPA commands, they are transferred using special short frames. All other commands except anticollision commands are transmitted using standard frames. Special frames called 'bit-oriented anticollision frames' are defined for the anticollision commands.

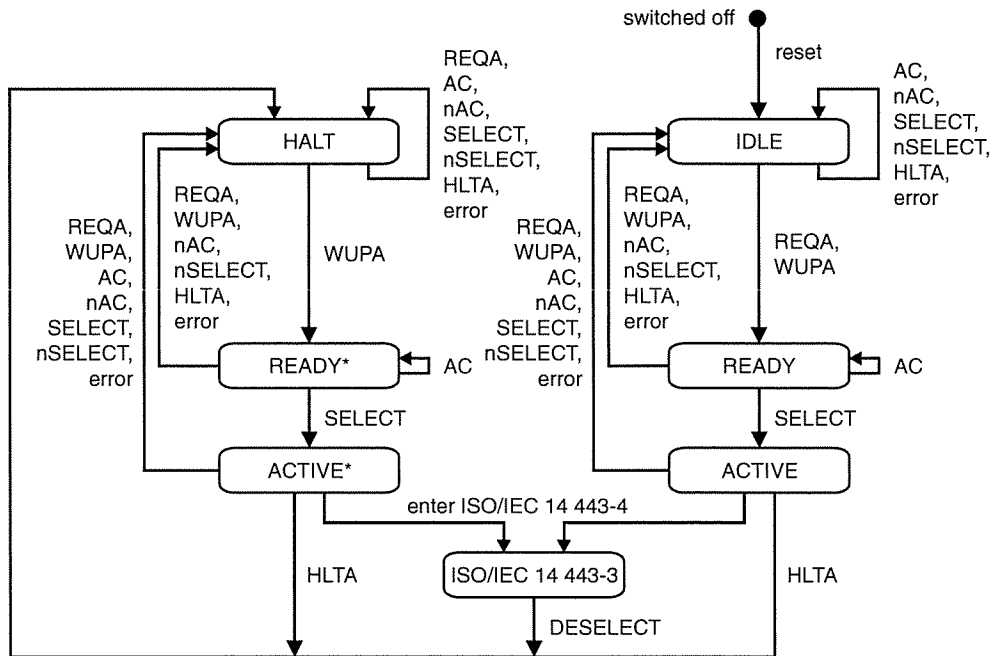


Figure 3.93 State diagram of a Type-A PICC during the initialization and anticollision phase. The abbreviations are explained in the text

Short frames

As already mentioned, short frames are only used for the initialization commands. A short frame consists of nine bits in the following sequence:

- one message start bit
- seven data bits starting with the least significant bit (lsb first)
- one message stop bit.

The coding rules for the start and end bits and the data bits are described in Section 3.6.3.1.

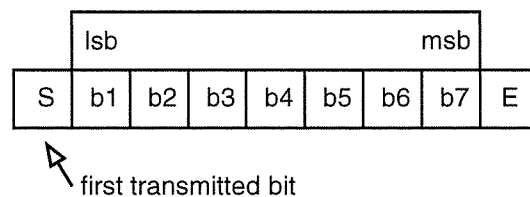


Figure 3.94 Structure of a short frame

Table 3.7 shows the coding of the REQA and WUPA commands, which are the only types of commands transmitted using short frames.

Table 3.7 Coding of the REQA and WUPA commands, which use the short-frame format with seven data bits

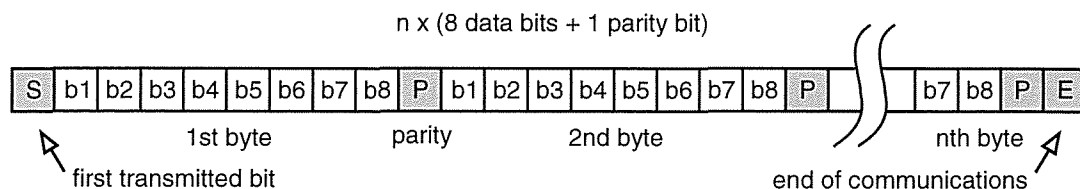
b7	b6	b5	b4	b3	b2	b1	Meaning
0	1	0	0	1	1	0	'26' = REQA
1	0	1	0	0	1	0	'52' = WUPA
0	1	1	0	1	0	1	'35' = optional timeslot method
1	0	0	x	x	x	x	'40' – '4F' = proprietary
1	1	1	1	x	x	x	'78' – '7F' = proprietary
all other values							RFU

The REQA and WUPA commands are transmitted by the terminal to determine whether any cards are present within the working range of the terminal (see Figure 3.93).

Standard frames

Standard frames are used for regular data exchanges. A standard frame consists of:

- start of message
- $n \times (8 \text{ data bits} + \text{odd-parity bit})$, with $n \geq 1$
- end of message.

**Figure 3.95** Structure of a standard frame

When the card changes to the Ready state, it transmits an Answer to Request, Type A (ATQA) after a precisely defined frame delay time (see Figure 3.98). An ATQA consists of two bytes, and due to the uniquely specified frame delay time, all ATQA messages are sent synchronously by all addressed cards. Figure 3.96 show the coding of the ATQA message.

When the terminal receives an ATQA, it recognizes that at least one card is present within its working range. It then initiates the anticollision procedure, which also allows it to read the Type-A unique identifier (UID), by transmitting a SELECT command. If the terminal is able to determine the complete identifier, it transmits a SELECT command containing this identifier. The card with the corresponding identifier confirms this command by transmitting a SELECT Acknowledge (SAK) message and changes to the Active state. In the Active state, the card can communicate using higher level protocols (such as those defined in ISO/IEC 14 443-4).

The card can be put into the Halt state by transmitting a HLTA command (Halt Command Type A). The card can also be put into the Halt state by means of special commands belonging to

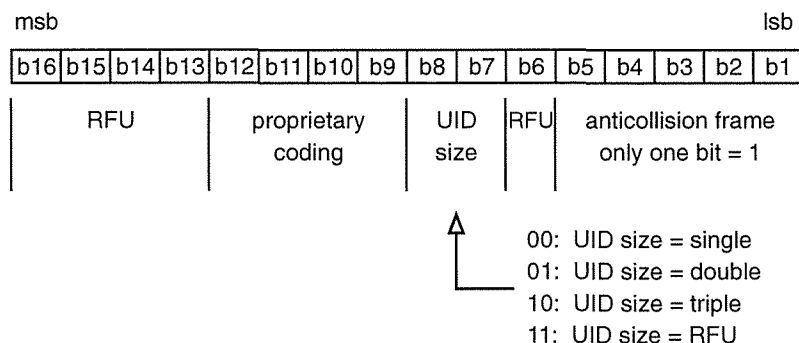


Figure 3.96 Coding of ATQA. All RFU bits must be set to 0. Bits 9–12 can be used to indicate other, non-standardized methods. One of bits b1–b5 must be set to 1. Bits b7 and b8 indicate the size of the UID

higher level protocols. In the Halt state, the card only responds to a WUPA (Wake-Up Type A) command, to which it responds by transmitting an ATQA (Answer to Request, Type A) and changing to the Ready* state. The Ready* state is similar to the Ready state. The conditions for changing to the Active* state are shown in Figure 3.93.

In detail, the procedure used for collision avoidance and determining the identifier works as follows. If two or more cards are concurrently in the Ready state and located within the working range of a terminal, they react simultaneously to a SELECT command from the terminal by each transmitting a portion of their different identifiers. This is done using a special bit-oriented frame, which allows the direction of data transmission between the terminal and the cards to be reversed after an arbitrary number of data bits have been transmitted. If several cards transmitting different data are present, the terminal will receive the data superimposed on each other, and it can detect a collision by the fact that this superimposition will cause the carrier to be modulated by the subcarrier for the full duration of one or more of the bit intervals. This is an irregular state, since the Manchester coding used requires a pulse edge to always occur within each bit interval. Figure 3.97 illustrates how this irregular state is produced.

In order for the terminal to be able to detect a collision at the bit level, all cards in the Ready state that are located within the working range of the terminal must respond to an ANTICOLLISION command at exactly the same time. To ensure this, the timing requirements imposed on the terminal and the card for exchanging frames are precisely specified in ISO/IEC 14 433-3.

Frame delay time (FDT)

The time between the end of the final pause transmitted by the terminal at the end of a message and the leading edge of the modulation pulse for the start bit transmitted by the card is designated the 'frame delay time PCD to PICC', which is abbreviated as FDT. This interval is defined in Figure 3.98. There are two different cases, depending on whether the final data bit transmitted by the terminal is a logic 1 or a logic 0.

For the REQA/WUPA, ANTICOLLISION and SELECT commands, the value of n is set to 9, which means that FDT is $1236/f_C$ or $1172/f_C$. This causes all cards within the working range of the terminal to respond synchronously to these commands, which are used in the

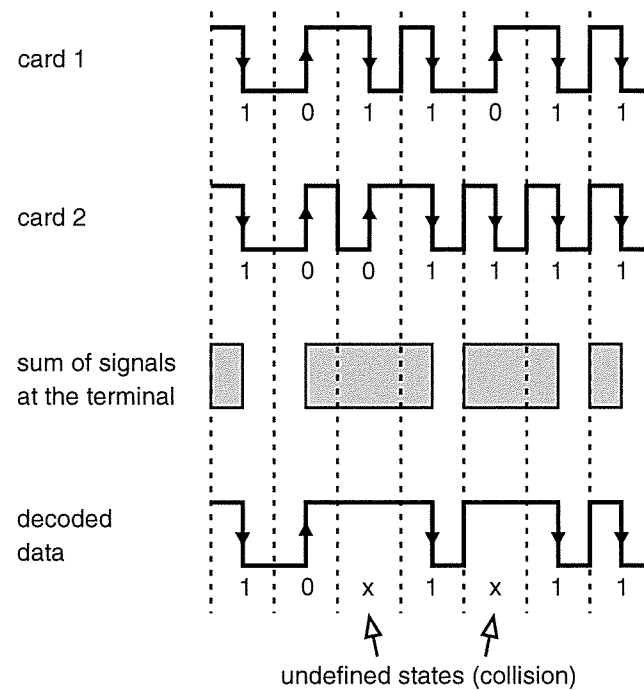


Figure 3.97 Collision of two bit sequences with Manchester coding (Type A). With an interference-free transmission, the carrier is always modulated by the subcarrier during only one half of each bit interval. If different bits are superimposed, modulation is present for the entire duration of the bit interval, allowing the terminal to detect a collision

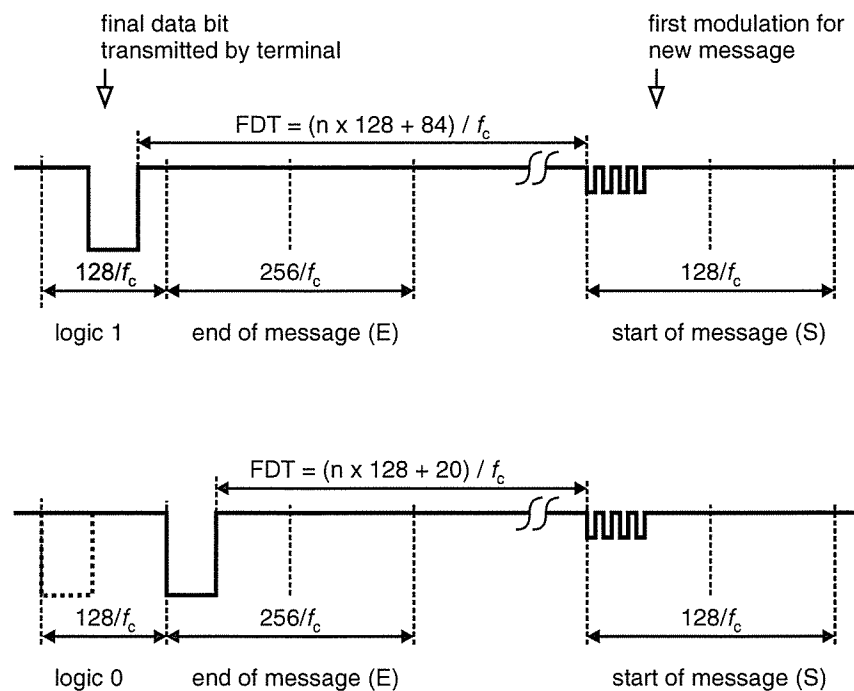


Figure 3.98 Frame delay time PCD to PICC (FDT)

anticollision loop. This makes it possible to detect collisions at the bit level. For all other commands, $n \geq 9$.

The time between the final modulation pulse transmitted by the card and the first pause signal transmitted by the terminal is designated the 'frame delay time PICC to PCD'. This time is not significant for collision detection at the bit level. Consequently, it is not precisely specified in the standard. The only requirement is that it must be equal to or greater than $1172/f_C$.

In addition, the minimum time between two successive REQA commands is specified to be $7000/f_C$. This time is designated the 'request guard time'.

As can be seen in the following sections, if there are several PICCs within the working range of the terminal, it may be necessary for the terminal to execute the anticollision loop many times before it can address a specific card. In order to avoid spending an excessive amount of time executing this loop, a special bit-oriented frame is used for the anticollision (AC) command, which is a special form of the SELECT command. This type of frame, which is called a 'bit-oriented anticollision frame', is described below.

Bit-oriented anticollision frame

If the terminal (PCD) detects a collision, it responds by transmitting bit-oriented anticollision frames, which have the same structure as standard seven-byte frames but are divided into two parts. The first part is used to transfer data from the PCD to the PICC, while the second part is used to transfer data in the opposite direction, from the PICC to the PCD. The relative lengths of the two parts vary in successive executions of the anticollision loop. The sum of the number of data bits in the two parts is always 56. The following rules apply to the lengths of the two parts:

- Rule 1: the total number of data bits is 56.
- Rule 2: the minimum length of Part 1 is 16 data bits.
- Rule 3: the maximum length of Part 1 is 55 data bits.

Consequently, Part 2 has a minimum length of one data bit and a maximum length of 40 data bits.

The frame can be divided between the two parts at any desired location, which means that the division may also be located within a data byte. In this case, no parity bit is appended to the first part of the divided byte, and the parity bit of the second part of the byte is ignored by the PCD. Two examples of anticollision frames are shown in Figure 3.99.

Commands used in the anticollision loop

As already indicated by the state diagram in Figure 3.93, the following commands may be used during initialization and in the anticollision loop:

- REQA (Request command, Type A)
- WUPA (Wake-Up command, Type A)

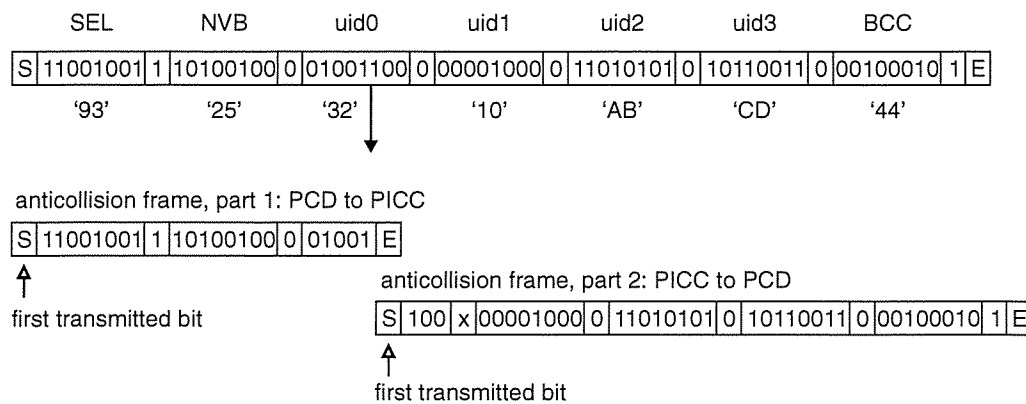


Figure 3.99 Two examples of bit-oriented anticollision frames. In the first example, the frame is divided after a full byte, while in the second example it is divided after the fifth bit of a data byte

- ANTICOLLISION
- SELECT
- HLTA (Halt command, Type A).

The ANTICOLLISION and SELECT commands

The ANTICOLLISION and SELECT commands are used in the anticollision loop. As can be seen in Figure 3.99, these commands consist of the following data fields:

- SEL (select code; 1 byte)
- NVB (number of valid bits; 1 byte)
- 0–40 data bits for the identifier (UID), depending on the value of NVB.

The unique identifier (UID) may consist of 4 bytes (single size), 7 bytes (double size) or 10 bytes (triple size).

Table 3.8 Possible sizes for the unique identifier (UID)

Cascade level	UID size	Number of UID bytes
1	Single	4
2	Double	7
3	Triple	10

The UID may be a fixed number or a random number generated by the card. With double- and triple-size identifiers, a cascade tag is transmitted as the first byte for cascade levels 1 and 2. The cascade tag is coded as '88'. In a single-size UID, the first byte is not allowed to have the value '88'.

Table 3.9 Coding of the SEL, which shows the current cascade level of the terminal

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
1	0	0	1	0	0	1	1	'93': cascade level 1 selected
1	0	0	1	0	1	0	1	'95': cascade level 2 selected
1	0	0	1	0	1	1	1	'97': cascade level 3 selected
1	0	0	1	all values other than those listed above				RFU

At most four data bytes can be transmitted from the PCD within an ANTICOLLISION or SELECT command (see Figure 3.99). If the card has a double- or triple-size identifier, it indicates this fact to the terminal in the SAK (Select Acknowledge) by setting the cascade bit, and it remains in the Ready state. The terminal will then start the anticollision procedure anew in order to ascertain bytes 5 through 7 of the UID. With a triple-size identifier, the anticollision procedure must be executed a third time to ascertain bytes 8 through 10. The terminal uses the SELECT command to inform the card of its current cascade level, in order to indicate which part of the UID is being requested (see Table 3.9).

This process is illustrated in Figure 3.100 in the form of a flow chart.

If the NVB byte does not specify 40 bits, the command is an ANTICOLLISION command and the card remains in the Ready or Ready* state. Once the terminal has ascertained the full UID, it sends a SELECT command with NVB = '70', which means that 40 data bits are specified. A CRC_A checksum formed in accordance with ISO/IEC 13 239 is appended to this command. The computation of this checksum is described in the informational Annex A of ISO/IEC 14 433-3 by means of an example. If the card is addressed using a SELECT command with its full identifier, it changes from the Ready state to the Active state, or from the Ready* state to the Active* state (see Figure 3.93), and then transmits SAK (Select Acknowledge) to indicate that the UID is complete. The codings of NVB (number of valid bits) and SAK are shown in Tables 3.10 and 3.11.

The procedure executed in the anticollision loop is shown in Figure 3.102 in the form of a flow chart for the terminal. This loop must be executed at every cascade level as long as the full UID is not known to the terminal.

To illustrate the selection process, Figure 3.103 shows an example in which two cards are within range of the terminal. In this example, card 1 (PICC 1) has a single-size UID with a value of '10' for uid0, while card 2 (PICC 2) has a double-size UID.

The terminal initiates the selection process by transmitting a REQA command. All cards within range of the terminal respond to this command. In this example, PICC 1 responds with an ATQA in which an anticollision bit frame is indicated by bit b1 being set and a single-size UID is indicated by bits b7 and b8 being cleared. PICC 2 also indicates an anticollision bit frame by setting bit b1 in its ATQA and indicates that it has a double-size UID by setting bit b7.

In the next step, the actual anticollision loop starts at cascade level 1. The terminal transmits an ANTICOLLISION command with a Select code of '93', which indicates an anticollision frame for cascade level 1. The value of '20' for NVB (number of valid bits) means that the terminal is not sending any portion of the cascade level 1 UID. Each card within range of

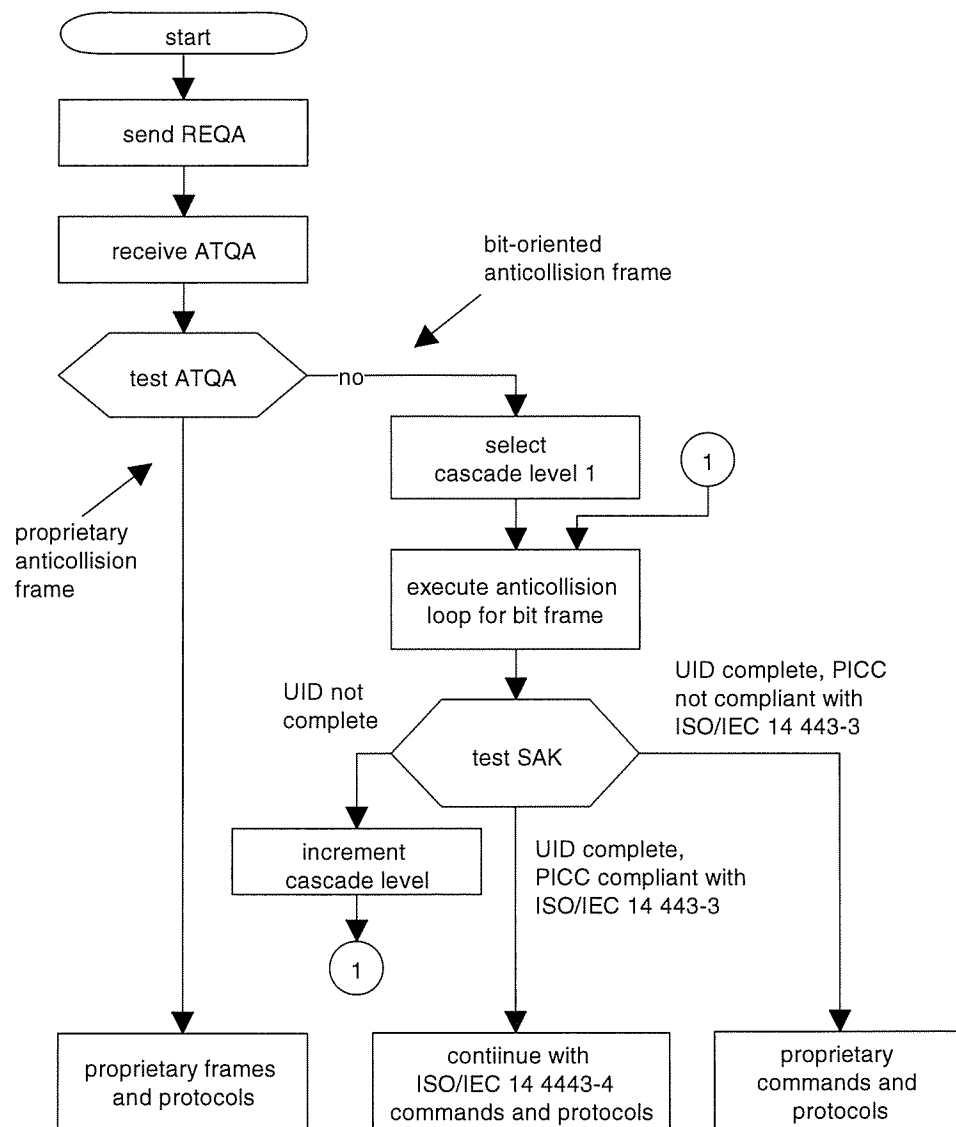


Figure 3.100 Flow chart for the process of selecting a particular card. After the terminal has transmitted a SELECT command, all cards in the Idle state respond simultaneously with ATQA

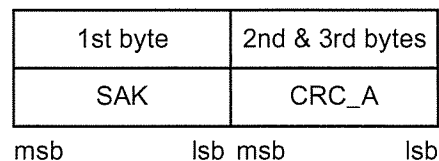


Figure 3.101 Format of SAK (Select Acknowledge). SAK is transmitted by the card when NVB indicates 40 data bits and these bits match the UID of the card

Table 3.10 Coding of NVB (number of valid bits). The upper four bits are called the 'byte count' and indicate the number of complete data bytes (including SEL and NVB) transmitted by the terminal. The lower four bits are called the 'bit count' and indicate the number of bits of any incomplete byte transmitted by the terminal

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
0	0	1	0	x	x	x	x	byte count = 2
0	0	1	1	x	x	x	x	byte count = 3
0	1	0	0	x	x	x	x	byte count = 4
0	1	0	1	x	x	x	x	byte count = 5
0	1	1	0	x	x	x	x	byte count = 6
0	1	1	1	x	x	x	x	byte count = 7
0	1	0	0	x	x	x	x	byte count = 8
x	x	x	x	0	0	0	0	bit count = 0
x	x	x	x	0	0	0	1	bit count = 1
x	x	x	x	0	0	1	0	bit count = 2
x	x	x	x	0	0	1	1	bit count = 3
x	x	x	x	0	1	0	0	bit count = 4
x	x	x	x	0	1	0	1	bit count = 5
x	x	x	x	0	1	1	0	bit count = 6
x	x	x	x	0	1	1	1	bit count = 7

Table 3.11 Coding of SAK (Select Acknowledge)

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
x	x	x	x	x	1	x	x	Cascade bit set:: UID incomplete
x	x	1	x	x	0	x	x	UID complete, PICC complies with ISO/IEC 14 443-4
x	x	0	x	x	0	x	x	UID complete, PICC does not comply with ISO/IEC 14 443-4

the terminal thus responds with its full cascade level 1 UID. The first bit collision occurs at bit b4. The terminal recognizes this bit collision and sends a new ANTICOLLISION command, this time containing the first three bits of the CL1 UID, which were received without errors, followed by a '1', and consequently with a value of '24' assigned to NVB. The first four bits now match the first four bits of the CL1 UID of card 2, but not the first four bits of the CL1 UID of card 1. Consequently, only card 2 responds with the remaining 36 bits of its CL1 UID. Since the terminal now knows the complete CL1 UID of card 2, it transmits a SELECT command for card 2, which responds with a SAK (Select Acknowledge) with the cascade bit (b3) set. From this, the terminal recognizes that the UID is not yet complete, so it increments the cascade level.

The terminal now sends a new ANTICOLLISION command with a Select code (SEL) indicating an anticollision bit frame and cascade level 2. NVB is set to '20' in order to request

Table 3.12 The steps of the anticollision algorithm. The numbering corresponds to Figure 3.102

Step 1	The terminal specifies the select code (SEL) for the cascade level.
Step 2	The terminal sets the NVB byte (number of valid bits) to '20'. This indicates that the terminal will not transmit any part of the UID, which means that every card within the working range of the terminal will transmit its complete CL _n UID (the portion of the UID for cascade level <i>n</i>).
Step 3	The terminal transmits an ANTICOLLISION command containing SEL and NVB.
Step 4	All cards within range of the terminal respond with their CL _n UIDs.
Step 5	If several cards with different identifiers are within range of the terminal, a collision will occur. If no collision occurs, steps 6 through 10 are skipped.
Step 6	The terminal determines the bit position of the first collision.
Step 7	The terminal assigns a value to NVB that indicates the number of valid bits of the CL _n UID. The valid bits are the portion of the CL _n UID that was received before the collision occurred, followed by a '0' or '1'.
Step 8	The terminal transmits an ANTICOLLISION command containing SEL, NVB and the valid bits.
Step 9	All cards whose corresponding portion of their CL _n UID matches the valid bits transmitted by the terminal transmit the remaining bits of their CL _n UIDs.
Step 10	If a collision occurs again, steps 6 through 9 are repeated.
Step 11	When no more collisions occur, the terminal sets NVB to '70'. This means that the terminal will transmit the complete CL _n UID.
Step 12	The terminal transmits a SELECT command containing SEL, NVB and the complete CL _n UID, followed by the CRC_A checksum.
Step 13	The card whose identifier matches the CL _n UID responds with SAK.
Step 14	If the UID is complete, the card transmits a SAK with the cascade bit cleared and changes from the Ready state to the Active state, or from the Ready* state to the Active* state.
Step 15	The terminal checks whether the cascade bit is set in order to decide whether an additional anticollision loop must be executed at a higher cascade level.

the complete CL2 UID from the card. Card 2 responds to this command with all 40 bits of its CL2 UID. The terminal can now transmit a SELECT command containing all of the bits of the CL2 UID, to which the card responds with a SAK in which cascade bit b3 is not set, indicating that its UID is complete. Card 2 now changes from the Ready state to the Active state, in which it can receive commands for higher level protocols.

Type-B initialization and anticollision

Type-B proximity cards use a 'dynamic slotted ALOHA procedure' for selection. In this procedure, cards within the working range of a terminal transmit their data to the terminal in predefined time slots. The time slots are selected randomly, and the number of slots can be determined by the terminal. If only a few slots are provided and significantly more cards are present in the working range of the terminal than the number of available slots, the probability that a card can transmit its data without interference is small. In such situations, interference-free data transmission can generally only occur if each time slot is used by only one card, or in

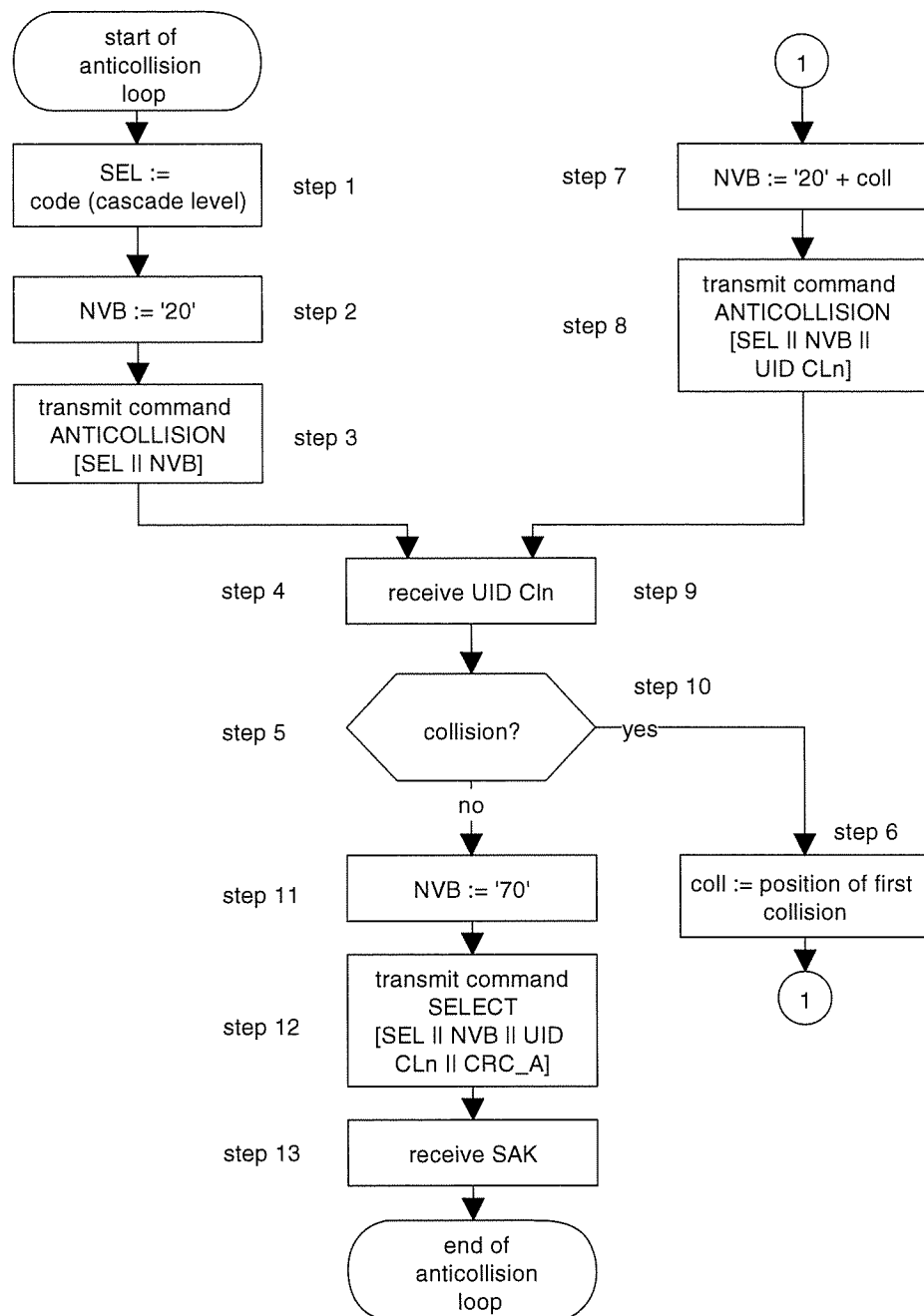


Figure 3.102 Flow chart of the anticollision loop as seen by the terminal. The step numbers ('Step 1', Step 2' etc.) refer to the algorithm steps listed in Table 3.12

the event that a slot is used by more than one card, if one of these cards is significantly closer to the terminal than the others, so the signal levels for its data prevail at the terminal. Although increasing the number of slots increases the probability of interference-free transmission, it has the disadvantage of increasing the amount of time required to execute the polling loop,

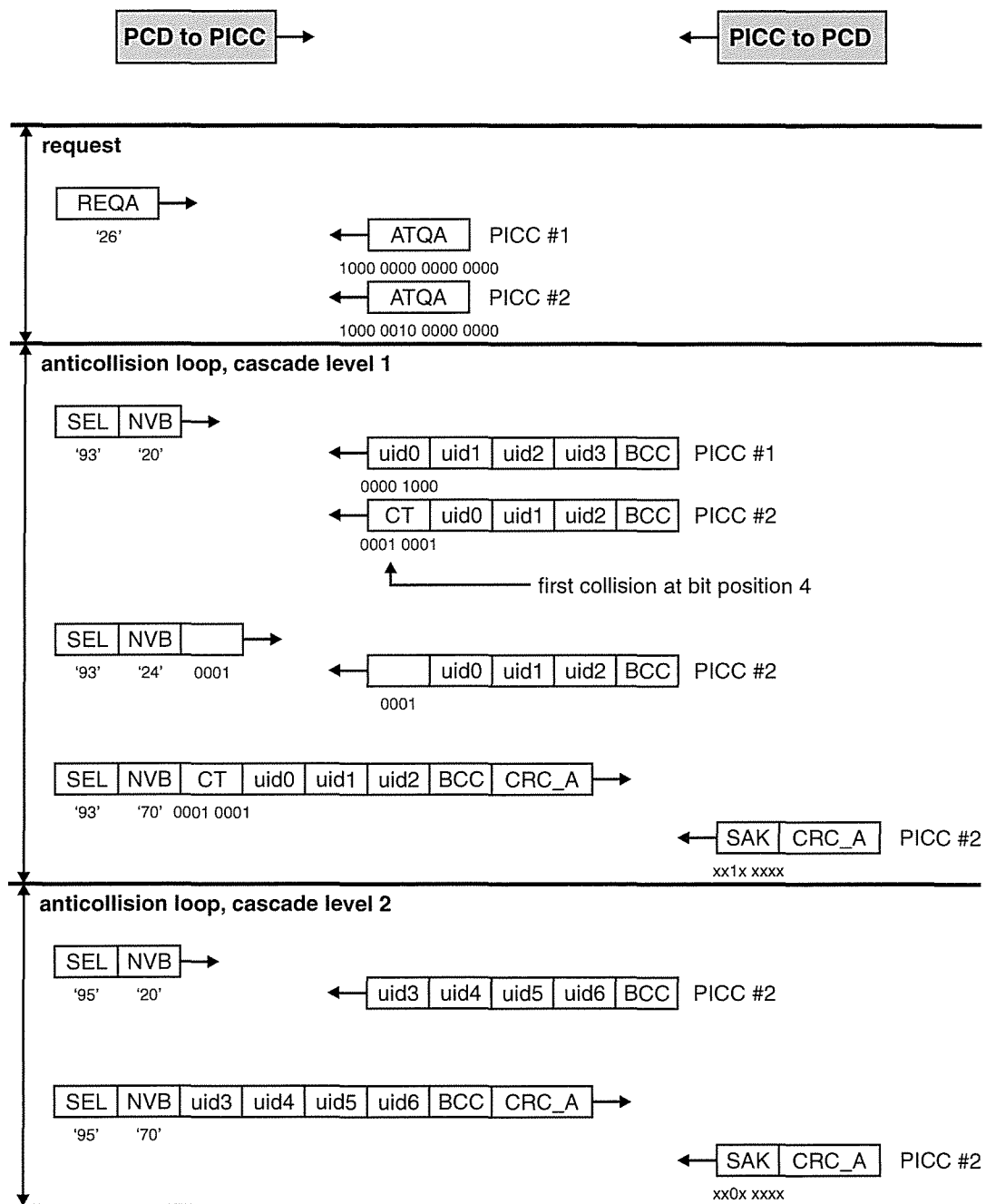


Figure 3.103 Sample initialization and anticollision sequence for Type-A cards. For the sake of simplicity, only the essential elements of the communications are shown

since each card must wait for the duration of all of the time slots provided for cards that might be within range of the terminal.

To make it easier to understand this procedure, which is specified in ISO/IEC 14 433-3, we will first explain it using a simplified example before describing it in detail. The terminal

needs two commands to execute this algorithm:

- **REQUEST** This command causes every card within the working range of the terminal to transmit its identifier in a subsequent time slot. In our example, four time slots will be provided.
- **SELECT** This command transmits a previously determined identifier to cards within the range of the terminal, causing the card with the matching identifier to be activated. Cards having other numbers remain passive and respond only to REQUEST or SELECT commands with matching identifiers.

When the terminal is in the operational state, it periodically transmits REQUEST commands. Here we assume that six cards having identifiers 1 through 6 are concurrently brought within the working range of the terminal. All six cards recognize the next REQUEST command, and they select time slots at random for transmitting their identifiers to the terminal (see Figure 3.104). In this example, collisions occur in time slots 1 and 3, while identifiers 2 and 3 are transmitted without interference in time slots 2 and 4. The terminal can now select one of these two cards using a SELECT command, and then communicate with the selected card without any further collisions.

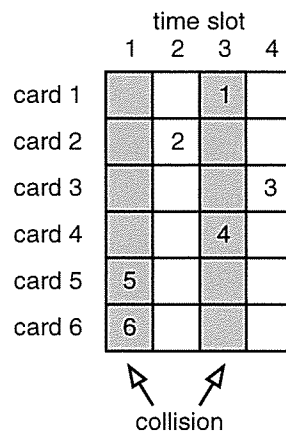


Figure 3.104 Example anticollision process for Type-B contactless smart cards. Collisions occur in time slots 1 and 3, so only identifiers 2 and 3 are transmitted without interference. This example is explained in detail in the text

When communication with the selected card is terminated, the terminal can search for other cards by again transmitting REQUEST commands. If it does not receive any error-free identifiers on the first attempt, it repeats the REQUEST command until it receives a valid identifier.

Now that the basic principle of this anticollision procedure has been explained, we can give our detailed attention to the commands and the timing behavior of the cards and the terminal, as specified in ISO/IEC 14 433-3 for Type-B cards. This standard defines a set of commands that allow various types of anticollision loops to be implemented. This gives users a certain amount of freedom for system optimization.

Formats and timing specifications for characters and frames

The terminal and the cards transmit data bytes in the form of characters, with several characters grouped into frames. For error detection, a 2-byte CRC checksum (CRC_B) computed according to ISOIEC 13 239 is appended to the characters in the frame. An example of the computation of CRC_B is given in Annex B of ISOIEC 14 443-3.

Each character consists of one start bit (logic 0) followed by eight data bits (with the least significant bit first) and one stop bit (logic 1).

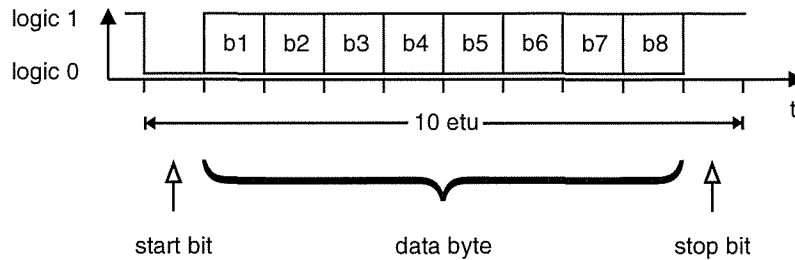


Figure 3.105 Character format

Each pair of characters is separated by a gap called the 'extra guard time' (EGT), which allows the transmitter and receiver to prepare for the next character. For data transmission to the card, this guard time ranges from 0 to 57 μs , while in the opposite direction it ranges from 0 to 19 μs .

Several characters are grouped together to form frames for transmission in each direction. A frame starts with a start of frame (SOF) character, followed by the characters to be transmitted and ending with an end of frame (EOF) character. The SOF character consists of a single falling edge followed by 10 etu of logic 0, with a rising edge in the eleventh etu, followed by a logic 1 for at least 2 and at most 3 etu.

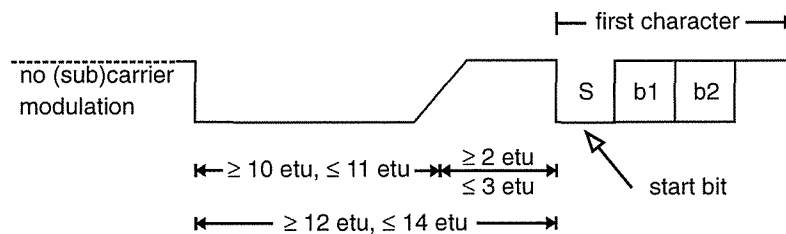


Figure 3.106 Timing of the start of frame (SOF) character

The EOF character also begins with a falling edge, followed by 10 etu of logic 0 and a rising edge within the eleventh etu.

To prevent the transmission protocol from 'hanging' in the event of an error, and to provide the card with defined minimum and maximum times for its internal activities, the times between two frames transmitted in opposite directions are specified in the standard. After the card has recognized the EOF of a frame transmitted by the terminal, it waits for the duration of the guard time (TR0) before generating the unmodulated subcarrier. After waiting for the duration of the synchronization time (TR1), the card starts to modulate the subcarrier. The minimum

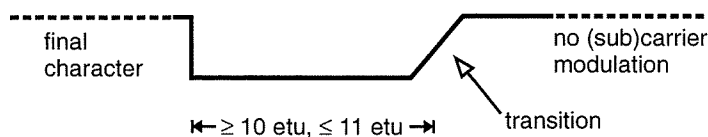


Figure 3.107 Timing of the end of frame (EOF) character

value for TR0 is $64/f_S$, while the maximum value in an anticollision loop (for an ATQB) is $256/f_S$. For all other types of frames, the maximum value is calculated using the formula

$$TR0_{\max} = (256/f_S) \times 2^{FWI}$$

The value of FWI ranges from 0 to 14 and is supplied to the terminal in the ATQB. The minimum value of the synchronization time (TR1) is $80/f_S$, and the maximum value is $200/f_S$.

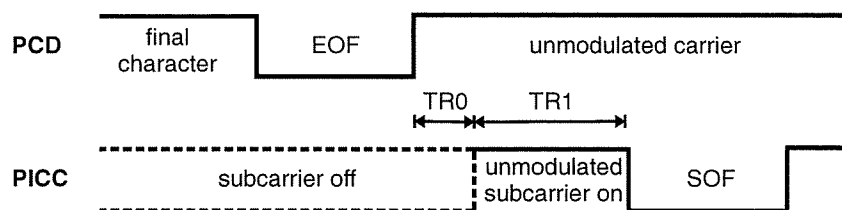


Figure 3.108 Definition of the guard time (TR0) and synchronization time (TR1)

Once the terminal recognizes the end of a frame transmitted by the card, it waits for an interval of at least $32/f_S$ before starting to send a new frame. During this interval, the card switches off the subcarrier within 2 etu of the end of the transmitted frame.

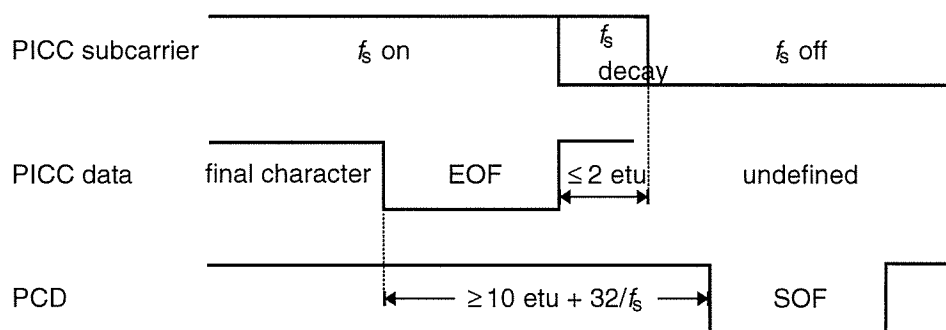


Figure 3.109 Definition of the waiting time between a frame transmitted by the card and the following frame transmitted by the terminal. The card switches off the subcarrier only after the end of the EOF character

Card selection procedure

If a Type-B card enters the working range of a terminal, the microprocessor in the card is supplied with power and enters the Idle state following the power-on reset. Just like Type-A

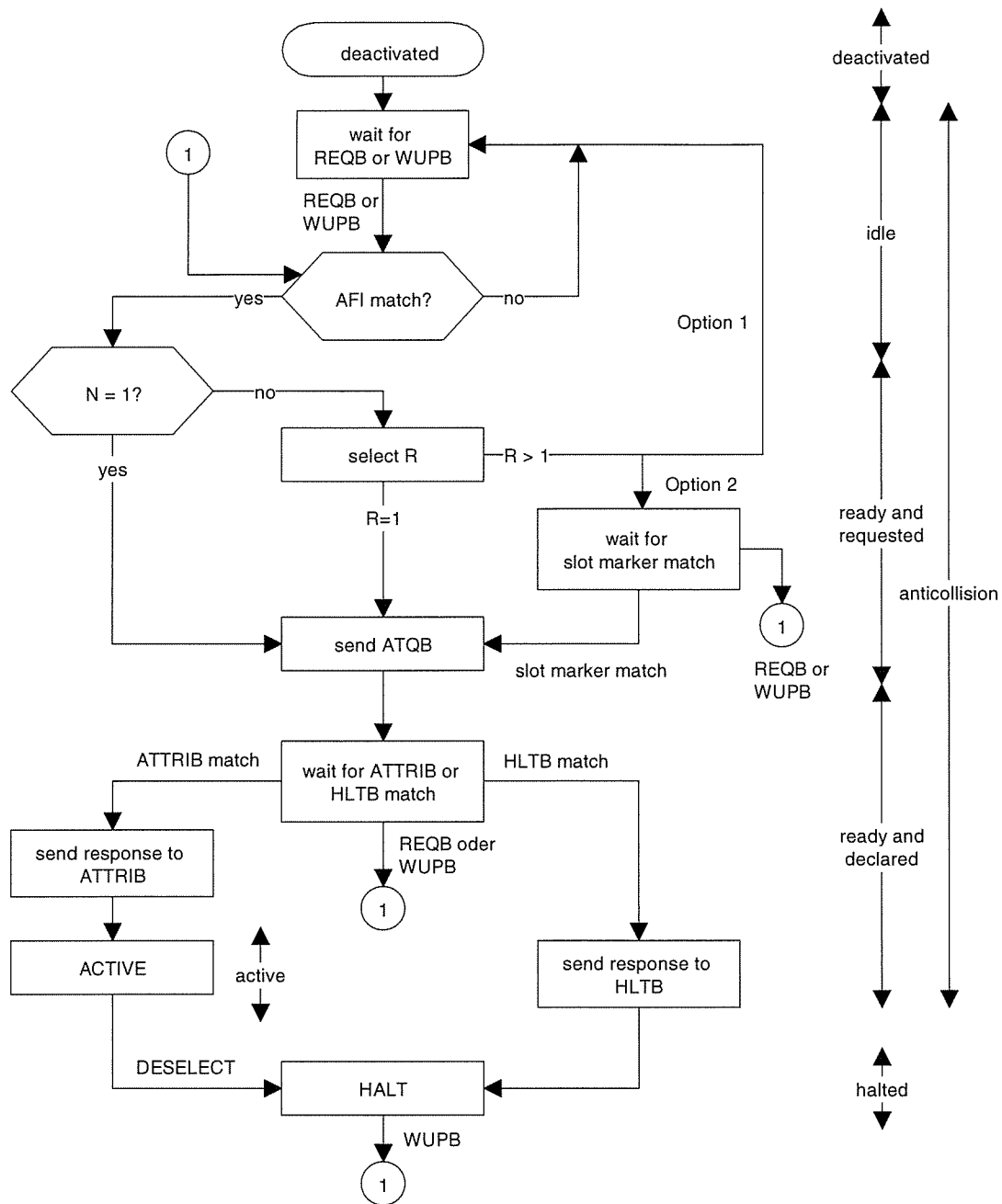


Figure 3.110 State diagram of a Type-B smart card according to ISO/IEC 14 443

cards, Type-B cards must enter the Idle state within 5 ms of receiving sufficient operating power from the field of the terminal. In the Idle state, the card is not allowed to send any frames. Instead, it must wait to receive a valid REQB or WUPB command. These commands contain a parameter indicating the number of time slots used by the terminal, along with an application family identifier (AFI) that can be used to prescribe a specific application group.

This produces a type of preselection, since only those cards whose applications belong to the indicated 'application family' will respond.

The REQB / WUPB command

The terminal transmits the REQB / WUPB command to determine whether any Type-B cards are present within its working range. The WUPB command is specifically used to wake up any cards that may be in the Halt state. A REQB / WUPB command consists of an anticollision prefix (APf) with a value of '05', followed by the AFI byte, a parameter byte (PARAM) indicating the number of available time slots and two CRC-B bytes.

APf	AFI	PARAM	CRC_B
1 byte	1 byte	1 byte	2 bytes

Figure 3.111 Format of the REQB/WUPB command

Table 3.13 Coding of the Application Family Identifier (AFI) byte

AFI upper nibble	AFI lower nibble	Meaning	Comments and sample applications
'0'	'0'	All families and subfamilies	No application preselection
X	'0'	All subfamilies of family X	Application preselection
X	Y	Only subfamily Y of family X	
'0'	Y	Only proprietary subfamily Y	
'1'	'0', Y	Transportation	Local public transport, busses, airlines, etc.
'2'	'0', Y	Financial sector	Electronic purses, banks, retail trade etc.
'3'	'0', Y	Identification	Access control, etc.
'4'	'0', Y	Telecommunications	Public telephone network, GSM, etc.
'5'	'0', Y	Medical	
'6'	'0', Y	Multimedia	Internet services, etc.
'7'	'0', Y	Games	
'8'	'0', Y	Data storage	Portable files, etc.
'9' – 'F'	'0', Y	RFU	

After a card has received a valid REQB command, it checks to see whether it supports the applications identified by the AFI. If it does, it evaluates the PARAM byte in order to obtain the value of N , which specifies the number of available slots. The card is now in the Ready Requested state.

If $N = 1$, the card transmits an ATQB (Answer to Request, Type B) and switches to the Ready Declared state. If $N > 1$, the card generates a random number R with a uniform distribution over the range of 1 to N . If $R = 1$, the card transmits an ATQB (Answer to

Table 3.14 Coding of the PARAM byte. Bit $b_4 = 0$ marks a REQB command (all cards in the Idle or Ready states respond to this command). Bit $b_4 = 1$ marks a WUPB command (all cards in the Idle, Ready or Halt states respond to this command)

b8	b7	b6	b5	b4	b3	b2	b1
RFU				REQB/WUPB	N (number of slots)		

Table 3.15 Coding of N (number of slots). The probability that a card responds in the first time slot is $1/N$. If only one time slot is used, the probability that a card responds in this slot depends on the value of N

b3	b2	b1	N
0	0	0	$2^0 = 1$
0	0	1	$2^1 = 2$
0	1	0	$2^2 = 4$
0	1	1	$2^3 = 8$
1	0	0	$2^4 = 16$
1	0	1	RFU
1	1	X	RFU

Request, Type B) and switches to the Ready Declared state. If $R > 1$, two different options are provided in the standard in order to support two different algorithms:

- Option 1:* This option is for cards that do not support selecting specific time slots. At this point, the card returns to the Idle state. It cycles through this loop repeatedly until $R = 1$ occurs by chance ('probabilistic approach') or the terminal sets the value of N to 1. This option does not actually use a time-slot method in the true sense, since only one time slot is used. This option is easy to implement and adequate for systems in which only a few cards are concurrently present within the working range of the terminal.
- Option 2:* This option is for cards that support time slot selection. In this case, the card waits until it receives a Slot Marker command with a matching time slot number (slot number = R) before transmitting an ATQB and switching to the Ready Declared state.

The Slot Marker command

The terminal sends a Slot Marker command at the start of each time slot. The format of this command is shown in Figure 3.112.

The coding of the 'anticollision prefix byte' (AP_n) is $AP_n = 'n5'$, where n is the number of the following slot. Slot Marker commands may be transmitted in any desired order; they do not have to be transmitted in increasing order of slot number.

After the Slot Number command has been transmitted, the terminal waits for an interval of $TR0_{\max} = 256/f_s$ before checking to see whether a card has started to transmit an ATQB.

AP_n	CRC_B
1 byte	2 bytes

Figure 3.112 Format of the Slot Marker command**Table 3.16** Coding of the slot number in anticollision prefix byte *n* (AP_n)

<i>n</i>	Slot number
0001	2
0010	3
0011	4
...	...
1110	15
1111	16

If the terminal does not detect an ATQB, it can immediately transmit the next Slot Marker command.

ATQB (Answer to Request, Type B)

The format of ATQB, which is sent in response a REQB/WUPB or Slot Marker command, is shown in Figure 3.113.

APa	PUPI	Application Data	Protocol Info	CRC_B
'50'	4 bytes	4 bytes	3 bytes	2 bytes

Figure 3.113 Format of ATQB (Answer to Request, Type B)

ATQB contains information regarding important parameters of the smart card, which the terminal needs in order to select the card. The pseudo-unique PICC identifier (PUPI) is the identification number of the PICC for the anticollision loop. This may be a number that is permanently assigned to the card, or it may be a random number generated by the card following power-on reset.

The Application Data field contains information about the applications present in the card. This information allows the terminal to select the desired card if several cards are present within its working range. The meaning of the application data parameter depends on the content of the application data coding (ADC) parameter in the Protocol Info field (described below), which specifies whether the 'CRC_B compressing method' or proprietary coding is used.

If the CRC_B compressing method is used, the Application Data field is formatted as shown in Figure 3.114.

The Protocol Info field shows important parameters supported by the card. These parameters allow the terminal to optimally adapt itself to the performance capacity of the card for the

AFI	CRC_B (AID)	Number of applications
1 byte	2 bytes	1 byte

Figure 3.114 Format of the Application Data field. The coding of the AFI parameter is given in Table 3.13. It indicates the family of the application for a multiapplication card. CRC_B (AID) is the ISO/IEC 7816-5-compliant CRC_B checksum of the application identifier (AID) of an application in the card corresponding to the AFI sent in the REQB / WUPB command. The Number of Applications field indicates the number of additional applications present in the card. The upper nibble of this byte indicates the number of applications corresponding to the AFI, where '0' means 'no applications' and 'F' means '15 or more applications'. The lower nibble indicates the total number of applications present in the card, with the same meaning ('0' = 'no applications', 'F' = '15 or more applications')

subsequent application protocol or adapt itself to cards that do not meet all the requirements of the standard.

Byte 1	Byte 2				
Bit_Rate_capability	Max_Frame_Size	Protocol_Type	FWI	ADC	FO
8 bits	4 bits	4 bits	4 bits	2 bits	2 bits

Figure 3.115 Format of the Protocol Info field

Tables 3.17 and 3.18 show the coding of the individual fields of the Protocol Info field.

Table 3.17 Coding of the FO (frame option) parameter

b2	b1	Meaning
1	x	PICC supports NAD
x	1	PICC supports CID

Table 3.18 Coding of the ADC (application data coding) parameter

b4	b3	Meaning
0	0	Proprietary application data coding
0	1	Application coded as described in the text

The frame waiting time integer (FWI) specifies the maximum amount of time needed by the card to start transmitting a response after it has fully received a command from the terminal. If a card does not respond within this interval, the terminal can assume that communications with the card have been interrupted. The frame waiting time (FWT) is calculated using the

following formula:

$$\text{FWT} = (256 \times 16/f_c) \times 2^{\text{FWI}}$$

The value of FWI lies between 0 and 14, with 15 being reserved for future use (RFU). The following minimum and maximum values for the frame waiting time can be calculated using this formula:

- minimum value (FWI = 0): $\text{FWT}_{\min} \approx 302 \mu\text{s}$
- minimum value (FWI = 14): $\text{FWT}_{\max} = 4949 \text{ ms}$

The Protocol_Type field indicates whether the card supports the ISO/IEC 14 443-4 transmission protocol. The coding of this field is shown in Table 3.19.

Table 3.19 Protocols supports by the card. All other values are reserved for future use (RFU)

b4	b3	b2	b1	Meaning
0	0	0	1	PICC supports ISO/IEC 14 443-4
0	0	0	0	PICC does not support ISO/IEC 14 443-4

In the Max_Frame_Size field, the card indicates the maximum frame size it can receive. This is limited by the size of the receive buffer in the card's RAM. Inexpensive chips typically have only a small amount of RAM, so they can only receive small frames. The Bit_Rate_capability field indicates the data transmission rates supported by the card, as shown in Table 3.21.

Table 3.20 Maximum frame size capacity of the card

Max_Frame_Size code in ATQB	0	1	2	3	4	5	6	7	8	9–F
Maximum frame size (bytes)	16	24	32	40	48	64	96	128	256	RFU > 256

Table 3.21 Bit rates supported by the card. All other values are reserved for future use (RFU)

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
0	0	0	0	0	0	0	0	PICC supports only 106 kbit/s in both directions
1	X	X	X	0	X	X	X	The same bit rate in both directions
X	X	X	1	0	X	X	X	PICC to PCD: 1 etu = $64/f_c$, 212 kbit/s supported
X	X	1	X	0	X	X	X	PICC to PCD: 1 etu = $32/f_c$, 424 kbit/s supported
X	1	X	X	0	X	X	X	PICC to PCD: 1 etu = $16/f_c$, 847 kbit/s supported
X	X	X	X	0	X	X	1	PCD to PICC: 1 etu = $64/f_c$, 212 kbit/s supported
X	X	X	X	0	X	1	X	PCD to PICC: 1 etu = $32/f_c$, 424 kbit/s supported
X	X	X	X	0	1	X	X	PCD to PICC: 1 etu = $16/f_c$, 847 kbit/s supported

As already mentioned, the card changes to the Ready Declared state after it transmits its ATQB (see Figure 3.111). In this state, the card responds only to the REQB/WUPB, ATTRIB and HALTB commands. It responds to a REQB/WUPB command in the same way as when it is in the Idle state.

If the card recognizes a valid ATTRIB command in which the PUPI matches the PUPI of the card, it transmits an Answer to ATTRIB frame and changes to the Active state. If the PUPI parameters do not match, the card remains in the Ready Declared state and waits for an ATTRIB command with the proper PUPI. The card responds to an appropriate HALTB command (containing the proper PUPI) by transmitting an Answer to HALTB and changing to the Halt state.

In the Active state, the card has a card identifier (CID) that is assigned to it by the ATTRIB command. As a result, it is in a higher protocol layer and responds to suitable application commands having the proper CID and correct CRC_B checksum. Special commands belonging to this higher protocol layer can put the card into the Idle or Halt state. When it is in the Active state, the card is not allowed to respond to REQB/WUPB, Slot Marker and ATTRIB commands.

In the Halt state, the card is passive and can only be reset to the Idle state by a valid WUPB command with the proper PUPI.

Format and coding of the ATTRIB command

The ATTRIB command is transmitted by the terminal to the card and contains information needed to select a card. It also contains information regarding the parameters supported by the terminal for subsequent communications and those required by the card for error-free communications. This includes parameters such as the minimum value of the guard time (TR0), the minimum value of the synchronization time (TR1), whether the card can suppress SOF and/or EOF to accelerate communications, the maximum frame size and selection of the optimum bit rate.

'ID'	Identifier (PUPI)	Param 1	Param 2	Param 3	Param 4	Higher Layer Inf.	CRC_B
1 byte	4 bytes	1 byte	1 byte	1 byte	1 byte	0 or more bytes	2 bytes

Figure 3.116 Format of the ATTRIB command. 'Identifier' contains the value of the PUPI, which is sent by the card in the ATQB. The format of Param 1 is shown in Figure 3.117

b8	b7	b6	b5	b4	b3	b2	b1
Minimum TR0		Minimum TR1		EOF	SOF	RFU	

Figure 3.117 Format of Parameter 1

The value of the 'Minimum TR0' parameter defines the minimum time that the card must wait before responding to a command received from the terminal. This is the time needed by the terminal to switch from transmit mode to receive mode, which depends on the performance of the terminal.

Table 3.22 Coding of the Minimum TR0 parameter

b8	b7	Minimum TR0
0	0	Default value
0	1	$48/f_s$
1	0	$16/f_s$
1	1	RFU

The value of the 'Minimum TR1' parameter defines the minimum delay between the activation of the subcarrier and the start of data transmission (see Figure 3.107). The terminal needs this time for synchronization with the card.

Table 3.23 Coding of the Minimum TR1 parameter

b6	b5	Minimum TR1
0	0	Default value
0	1	$64/f_s$
1	0	$16/f_s$
1	1	RFU

Bits b3 and b4 indicate to the terminal whether the card supports suppression of EOF and/or SOF from the card to the terminal in order to reduce communications overhead. This capability is optional for the card.

Table 3.24 SOF utilization

b3	SOF required
0	yes
1	no

Table 3.25 EOF utilization

b4	EOF required
0	yes
1	no

The lower nibble of Parameter 2 (bits b4–b1) specifies the maximum size of a frame that can be received from the terminal. The upper nibble is used to select the bit rates in both directions. The terminal can make this choice, since it already knows the bit rates supported by the card from the ATQB.

The lower nibble of Parameter 3 is used to confirm the protocol type. The coding corresponds to that shown in Table 3.19. The upper nibble is set to '0'. All other values are reserved for future use.

Table 3.26 Coding of bits b4–b1 in Parameter 2, which specify the maximum frame size

Max_Frame_Size code in ATTRIB	0	1	2	3	4	5	6	7	8	9–F
Maximum frame size (bytes)	16	24	32	40	48	64	96	128	256	RFU>256

Table 3.27 Coding of bits b8–b5 in Parameter 2, which select the transmission bit rate

b8	b7	b6	b5	Meaning
0	0	x	x	PICC to PCD: 1 etu = $128/f_C$, bit rate is 106 kbit/s
0	1	x	x	PICC to PCD: 1 etu = $64/f_C$, bit rate is 212 kbit/s
1	0	x	x	PICC to PCD: 1 etu = $32/f_C$, bit rate is 424 kbit/s
1	1	x	x	PICC to PCD: 1 etu = $16/f_C$, bit rate is 847 kbit/s
x	x	0	0	PCD to PICC: 1 etu = $128/f_C$, bit rate is 106 kbit/s
x	x	0	1	PCD to PICC: 1 etu = $64/f_C$, bit rate is 212 kbit/s
x	x	1	0	PCD to PICC: 1 etu = $32/f_C$, bit rate is 424 kbit/s
x	x	1	1	PCD to PICC: 1 etu = $16/f_C$, bit rate is 847 kbit/s

Parameter 4 also consists of two parts. The lower nibble is called the ‘card identifier’ (CID) and defines the logical number of the addressed card, with a range of 0 to 14. The value 15 is reserved for future use. The card identifier is specified by the terminal and is unique for each active card. If the card does not support CID, a value of ‘0’ is used. The upper nibble is set to ‘0’. All other values are reserved for future use.

The ‘Higher-Layer Inf’ field can be used to transfer any desired higher level commands. The ability to process such commands is optional for the card.

Response to the ATTRIB command

The card responds to every valid ATTRIB command (having the proper PUPI and correct CRC_B checksum) as shown in Figure 3.118.

Byte 1		Bytes 2–n	
MBLI	CID	Higher-layer response	CRC_B
1 byte		Optionally 0 or more bytes	2 bytes

Figure 3.118 Format of the response to an ATTRIB command

If the terminal receives a valid response to an ATTRIB command (one having the same CID and a correct CRC_B checksum), it knows that card selection was successful. The lower nibble of the first byte in the response (bits b4–b1) contains the CID. The upper nibble of the first byte (bits b8–b5) is called the ‘maximum buffer length index’ (MBLI). The card uses the MBLI to tell the terminal the maximum size of its input buffer. This allows the terminal to avoid causing the input buffer of the card to overflow by sending too many chained frames. If

MBLI is set to 0, the card does not provide any information about the size of its internal buffer. If MBLI is greater than 0, the maximum internal buffer length (MBL) can be calculated using the following formula:

$$\text{MBL} = (\text{PICC maximum frame size}) \times 2^{(\text{MBLI}-1)}$$

The card sends its maximum frame size parameter to the terminal in the ATQB. When the terminal transmits chained frames, it must ensure that the cumulative length does not exceed the value of MBL.

The HLTB command

The HLTB command is used to place a card in the Halt state, so that it no longer responds to REQB commands. After responding to this command, the card ignores all subsequent commands except WUPB.

'50'	Identifier	CRC_B
1 byte	4 bytes	2 bytes

Figure 3.119 Format of the HLTB command

The Identifier parameter contains the PUPI of the card to be placed in the Halt state. The format of the card's response to a valid HLTB command is shown in Figure 3.120.

'00'	CRC_B
1 byte	2 bytes

Figure 3.120 Format of the response to a HLTB command

Example of an anticollision sequence with three Type-B cards

The standard gives the developer the freedom to implement various types of anticollision strategies. This corresponds to the basic function of a standard, which is to make interoperability possible while providing as much latitude as possible for implementation in order to avoid hindering technical progress. An example of an anticollision sequence is shown in Figure 3.121. This example, which is also contained in the annex to the standard, serves to illustrate the processes and commands described in the previous section. It makes no claim to being a technically superior implementation.

3.6.3.4 Data transmission protocol (ISO/IEC 14 433-4)

A half-duplex block transmission protocol that is tailored to the specific requirements of a contactless system is defined in Part 4 of ISO/IEC 14 433-4. This protocol is largely based on

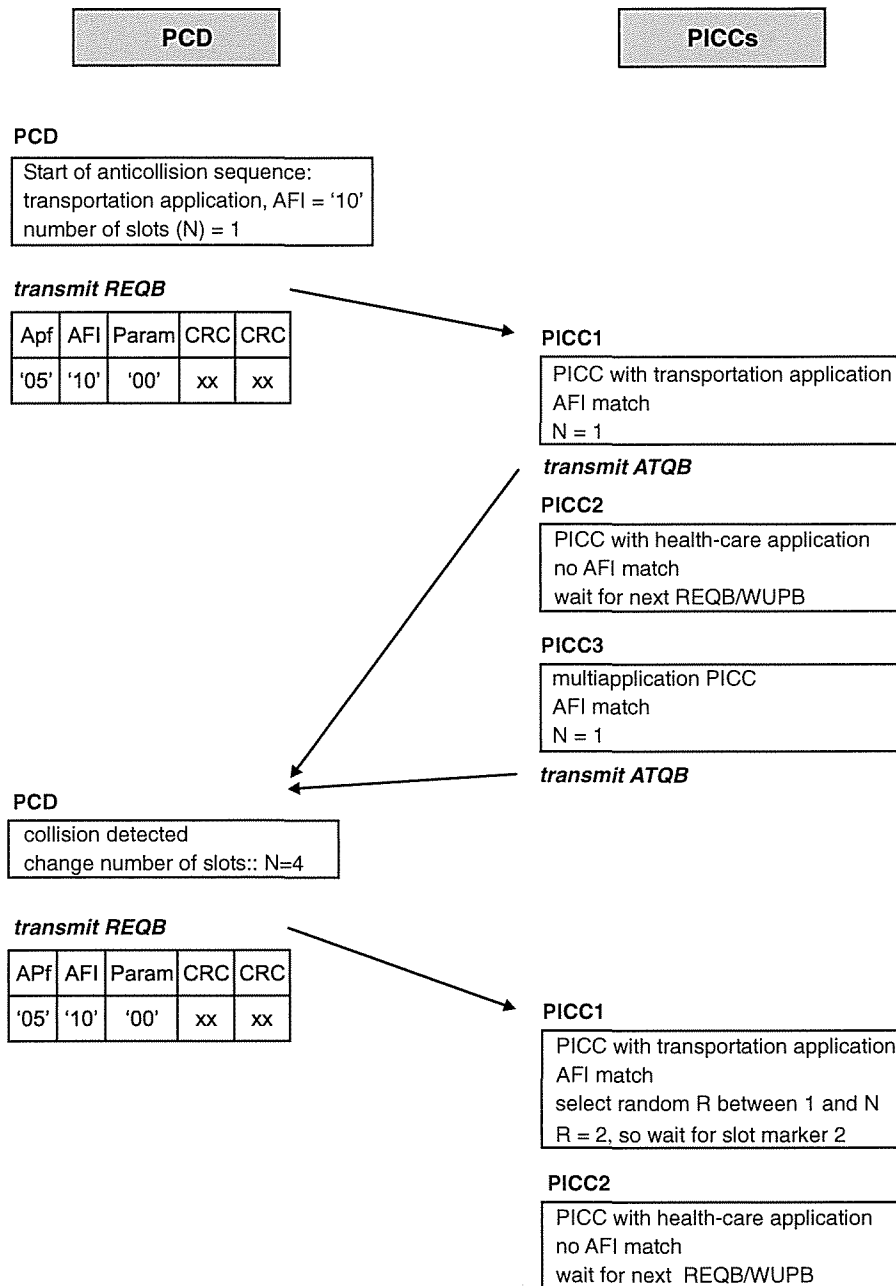


Figure 3.121 Example of an anticollision sequence with three Type-B cards (Part 1 of 2)

the T = 1 protocol defined in the ISO/IEC 7816-3 standard, which is widely used throughout the world. This simplifies the implementation of dual-interface cards, since they anyhow must support a contact-type transmission protocol.

For Type-A cards, the standard defines an activation sequence that must be executed before starting the actual protocol. During this sequence, parameters for the subsequent data transmission are specified and exchanged between the terminal and the card. These parameters

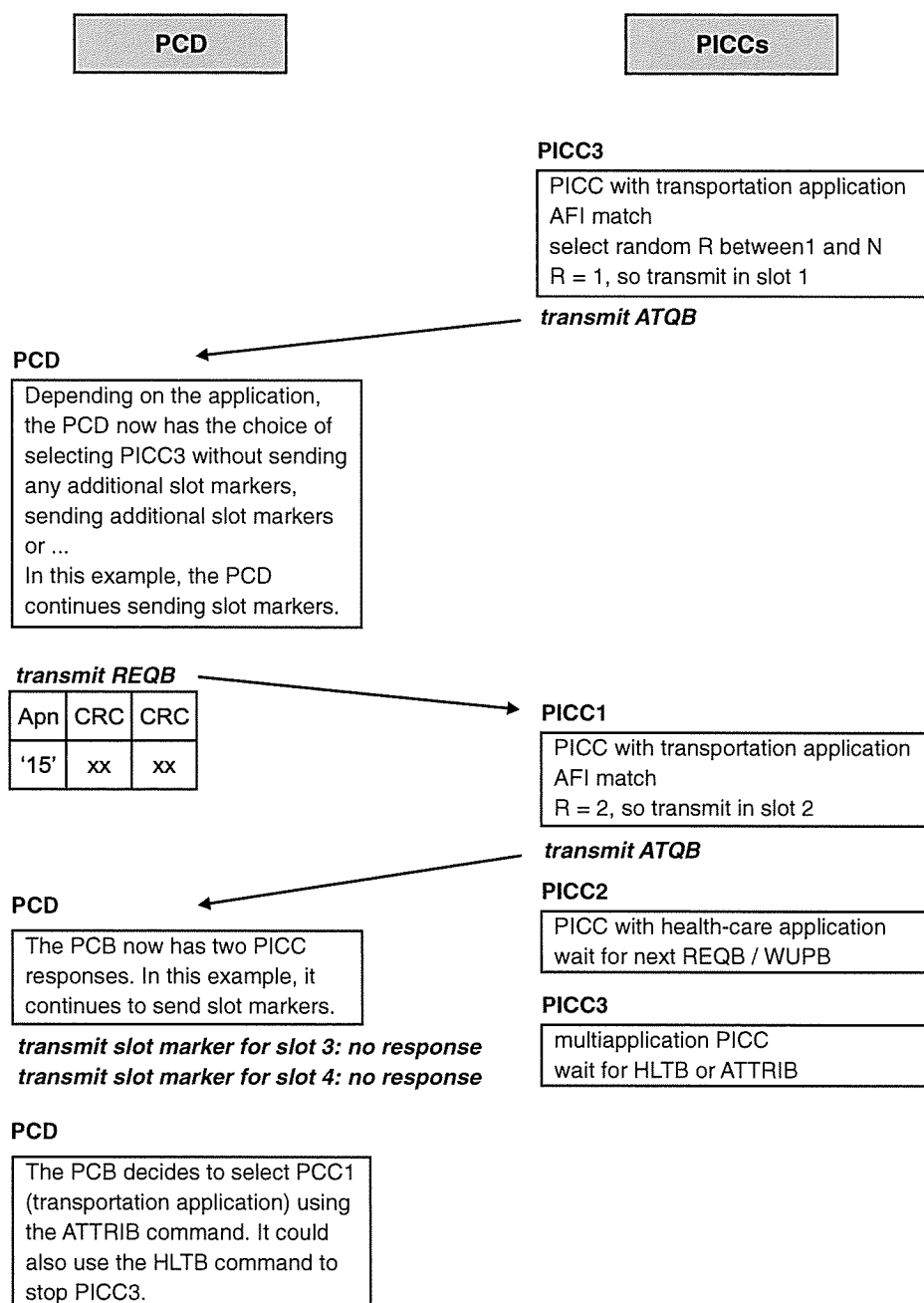


Figure 3.121 Example of an anticollision sequence with three Type-B cards (Part 2 of 2)

relate to things such as the bit rate in each direction and the required waiting times between frames.

Type-B cards do not need any special activation sequence. They can immediately initiate the actual data transmission protocol after being selected. With such cards, the necessary parameters for data transmission are specified and exchanged during the initialization and selection processes, as described in the previous section.

Protocol activation for Type-A cards

After a Type-A PICC has been successfully selected, as previously described, the terminal executes an activation sequence as illustrated by the flow chart shown in Figure 3.122.

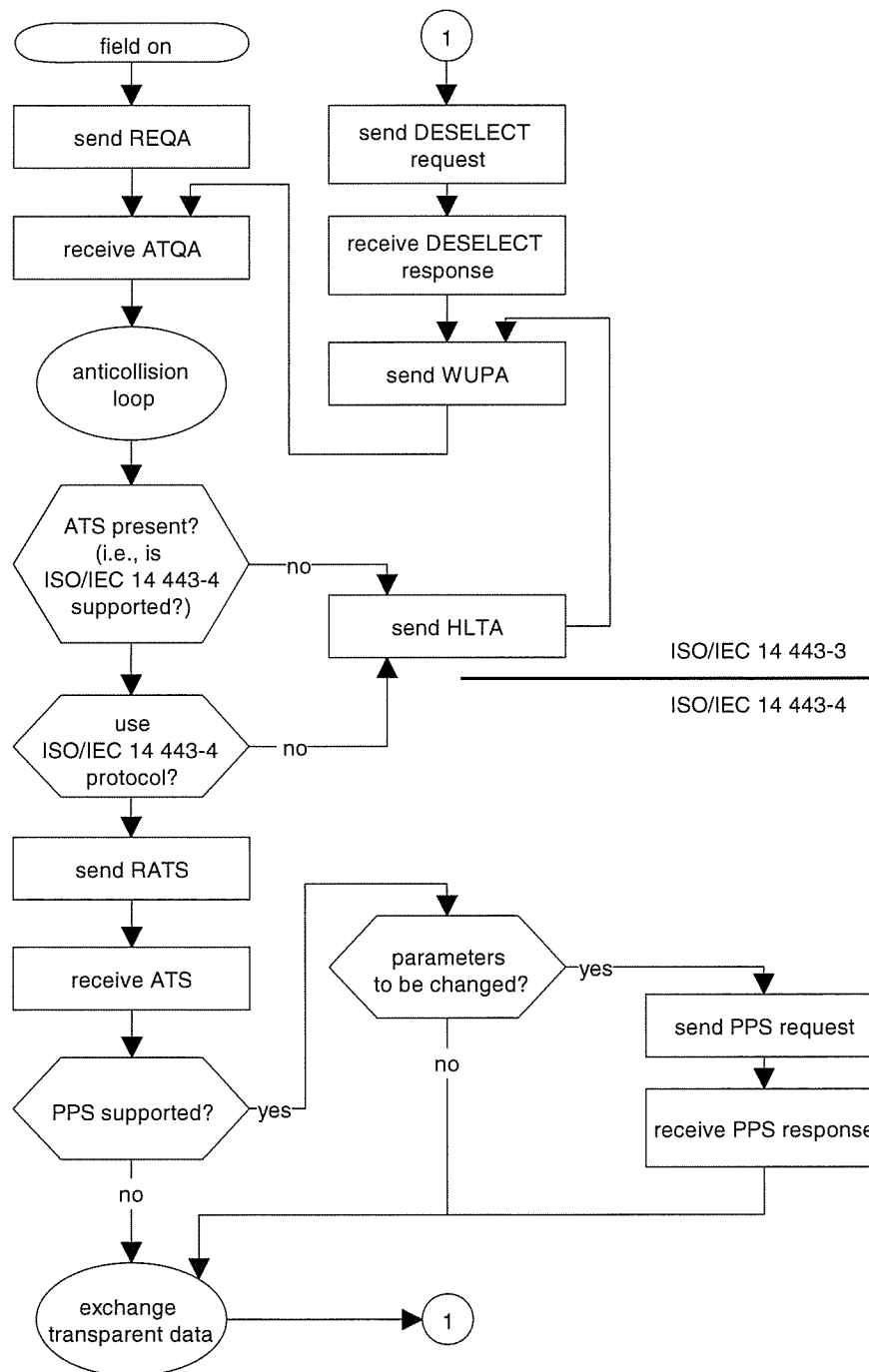


Figure 3.122 Activation of a Type-A card by a terminal

From the SAK (Select Acknowledge) transmitted by the card at the end of the anticollision loop, the terminal can recognize whether the card supports the standard data transmission protocol. If it does not, the terminal issues a HLTA command to place the card in the Halt state. If the protocol defined in ISO/IEC 14 433-4 is supported, the terminal sends a RATS (request for answer to select) command to the card. The RATS command and the ATS (answer to reset) returned by the card are used to exchange data and parameters in order to determine the data transmission options supported by the card and the terminal. Following this, the values of the modifiable parameters can be selected using PPS (protocol and parameter selection) to make optimum use of the capabilities of the card and the terminal. In order to make inexpensive, technically simple implementations possible, default values are defined for the modifiable parameters. In the simplest case, the card supports only these default values. In this case, the PPS sequence is unnecessary, and the terminal can immediately start exchanging data using the block protocol after receiving the ATS.

Request for answer to select (RATS)

The RATS command contains a parameter byte specifying the maximum frame size that the terminal can receive ('frame size for proximity coupling card', or FSDI) and the card identifier (CID) assigned to the card for the duration of its active state. Starting with the reception of the RATS command, the card uses this CID as its logical identifier.

'E0'	Parameter	CRC_A
1 byte	1 byte	2 bytes

Figure 3.123 Format of the RATS (Request for Answer to Select) command

b8	b7	b6	b5	b4	b3	b2	b1
FSDI				CID			

Figure 3.124 Format of the Parameter byte of the RATS command. The CID defines the logical number of the addressed card and has a range of 0 through 14; 15 is reserved for future use. FSDI codes the maximum frame size (FSD) that the terminal can receive

Table 3.28 Coding of bits b8–b5 of the Parameter byte of the RATS command

FSDI	0	1	2	3	4	5	6	7	8	9–F
FSD (bytes)	16	24	32	40	48	64	96	128	256	RFU>256

Answer to Select (ATS)

A Type-A card responds to a RATS command with an Answer to Select (ATS). The ATS identifies the set of parameters supported by the card. These parameters may include:

- the maximum frame size
- the bit rates in both directions supported by the card
- the waiting time between frames
- the specific frame guard time
- support for NAD and CID.

Default values are specified for cards that do not offer any selection of parameters. In the simplest case, which is when only the default values are supported, the ATS consists of only the length byte and the CRC bytes.

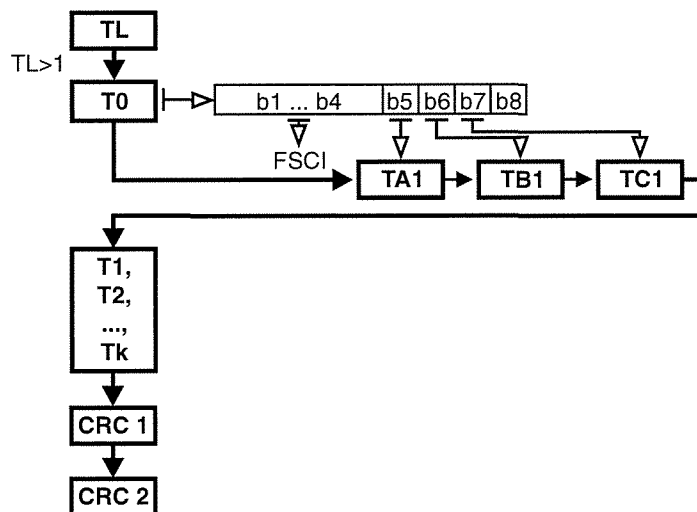


Figure 3.125 Format of ATS

Table 3.29 The data elements of the ATS and their designations according to ISO/IEC 14 433-4

Data element	Designation
TL	Length byte
T0	Format byte
TA1, TB1, TC1	Interface bytes
T1, T2, ... Tk	Historical bytes
CRC1, CRC2	Cyclic redundancy check

Length byte The length byte (TL) indicates the number of bytes transmitted in the ATS, including the TL byte but excluding the two CRC bytes. The length of the ATS is not allowed

to exceed the maximum frame length (FSD) given in the RATS command. This means that the maximum value of TL is not allowed to be greater than $FSD - 2$.

Format byte If the length given in TL is greater than 1, the format byte (T0) is sent next. T0 consists of three parts, as follows:

- The most significant bit (b8) has a value of 0. The value 1 is reserved for future use.
- Bits b7–b5 indicate the presence of subsequent interface bytes TC1, TC2 and TC3.
- The lower nibble (b4–b1) is called the ‘frame size for proximity card integer’ (FSCI). It codes the value of ‘frame size for proximity card’ (FSC), which is the maximum frame size that can be received by the card. The default value for FSCI is 2, which corresponds to a maximum frame size of 32 bytes.

Table 3.30 Coding of the format byte (T0)

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
0	...	...	...	...	...	...	...	Set to 0 (1 is RFU)
0	1	X	X	...	...	...	...	TC1 transmitted
0	X	1	X	...	...	...	...	TB1 transmitted
0	X	X	1	...	...	...	...	TA1 transmitted
0	...	...	...	X	X	X	X	Maximum frame size (FSCI)

Table 3.31 Coding of bits b4–b1 of the FSCI parameter

FSCI	0	1	2	3	4	5	6	7	8	9–F
FSC (bytes)	16	24	32	40	48	64	96	128	256	RFU>256

TA1 interface byte The TA1 interface byte consists of four parts, as follows:

- The most significant bit (b8) indicates whether different divider values can be used for the two transmission directions. The value of the etu (elementary time unit, equal to the duration of one bit) is determined by the divider D according to the formula $1 \text{ etu} = 128 \div (D \times f_C)$. The initial value of D is 1, which yields an etu value of $128/f_C$.
- Bits b7–b5, which are called ‘divisor send’ (DS), specify the bit rates supported by the card for data transmission from the card to the terminal.
- Bit b4 is set to 0. The value 1 is RFU.
- Bits b3–b1, which are called ‘divisor receive’ (DR), specify the bit rates supported by the card for data transmission from the terminal to the card.

The divider values are selected by the terminal in the subsequent PPS command.

Table 3.32 Coding of the TA1 interface byte

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
0	...	...	...	0	...	...	...	Different values of D are supported for data transmissions in the two directions
1	...	...	...	0	...	...	...	Only one value of D is supported for data transmissions in the two directions
...	1	...	...	0	...	...	...	DS = 8 is supported
...	...	1	...	0	...	...	...	DS = 4 is supported
...	...	...	1	0	...	...	...	DS = 2 is supported
...	...	...	...	0	1	...	...	DR = 8 is supported
...	...	...	...	0	...	1	...	DR = 4 is supported
...	...	...	...	0	...	...	1	DR = 2 is supported

TB1 interface byte The TB1 interface byte is used to transfer parameters that define the frame waiting time and the start-up frame guard time. Consequently, the TB1 interface byte consists of two parts, as follows:

- The upper nibble (b8–b5) is called the ‘frame waiting time integer’ (FWI) and determines the frame waiting time (FWT). The meaning and calculation of the frame waiting time are described in the next section.
- The lower nibble (b4–b1) is called the ‘start-up frame guard time integer’ (SFGI) and is used to calculate the start-up frame guard time (SFGT). The SFGT is the time needed by the card after transmission of the ATS before it is ready to receive the next frame. The value 15 is RFU. The value 0 means that the card does not need any particular SFGT. With a value ranging from 1 through 14, the SFGT is calculated using the formula:

$$\text{SFGT} = (256 \times 16/f_C) \times 2^{\text{SFGI}}$$

The default value of SFGI is 0, and its maximum value (SFGT_{MAX}) is approximately 4949 ms.

b8	b7	b6	b5	b4	b3	b2	b1
FWI				SFGI			

Figure 3.126 Format of the TB1 interface byte

TC1 interface byte The TC1 interface byte indicates special protocol options. It consists of the following parts:

- The six most significant bits (b8–b3) are set to '0'. All other values are RFU.
- Bits b2 and b1 indicate which optional fields in the prologe field are supported by the card (see the next section). The terminal is not obliged to actually transmit all fields that are supported, so it may omit one or more fields. However, fields that are not supported by

the card may not be transmitted by the terminal to the card under any circumstances. The default value for bit b2 is 1, and the default value for b1 is 0. These values mean that CID is supported and NAD is not supported.

Table 3.33 Coding of the TC1 interface byte

b8	b7	b6	b5	b4	b3	b2	b1
0	0	0	0	0	0	b2 = 1 means that CID (card identifier) is supported	b1 = 1 means that NAD (node address) is supported

Historical bytes The historical bytes (T1 through Tk) are optional. Their contents are defined in ISO/IEC 7816-4 (see also Section 6.2, ‘Answer to Reset (ATR)’). The maximum possible number of historical bytes can be determined from the maximum length of the ATS.

Protocol and parameter selection

If the card indicates in the ATS that it supports modifiable parameters, the terminal can change the parameters for the subsequent protocol by using the PPS (protocol and parameter selection) command. If the card does not support any modifiable parameters, it is not required to support the PPS command (see Figure 3.122). In this case, the protocol will continue with unaltered parameters.

If we assume that the card has indicated in the ATS that it supports modifiable parameters, the terminal can now evaluate whether it wants to utilize the modification options indicated by the card. If so, it transmits a PPS Request command having the structure shown in Figure 3.127.

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5
PPSS	PPS0	PPS1	CRC1	CRC2
start byte	Parameter 0 indicates whether PPS1 is present	Parameter 1 codes DRI and DSI		

Figure 3.127 Format of the PPS Request (protocol and parameter selection request) command

Start byte The start byte (PPSS) consists of two parts:

- The upper nibble (b8–b5) is set to 'D' to identify the PPS. All other values are RFU.
- The lower nibble (b4–b1), which is called the ‘card identifier’ (CID), defines the logical number of the addressed card.

Parameter 1 Parameter 1 indicates which of the possible values of DS and DR in the TA1 interface byte have been selected. This determines the transmission rates for subsequent data transmissions.

Table 3.34 Coding of Parameter 1 (PPS1)

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
0	0	0	0	DSI		DRI		DRI and DSI code the divisor D

Table 3.35 Coding of D by DRI and DRS

DRI / DSI	0	0	0	1	1	0	1	1
D	1		2		4		8	

Protocol and parameter selection response

The card confirms correct reception of the protocol and parameter selection request by means of a protocol and parameter selection response, which consists of only the PPSS start byte and the two checksum bytes (CRC 1 and CRC 2). After this, the terminal and the card use only the selected parameters for data transmission.

Activation frame waiting time

In order to avoid unnecessarily long waiting times in case of transmissions errors, ISO/IEC 14 433-4 specifies the maximum time between when a Type-A card receives the end of a frame and when it sends a response. This time is called the ‘activation frame waiting time’, and it is set to $65,536/f_C$ ($\approx 4833 \mu\text{s}$). If a card does not respond within this interval, the terminal knows that communications with the card are impaired.

Error detection and correction

In a system using contactless cards, it must be expected that errors will occur more often during data transmission than is usual with contact-type cards. For example, with contact-type cards it is relatively uncommon for a card to be removed from a terminal while it is communicating with the terminal. With contactless cards, interruptions to communications can occur more often, since the cards are free to move within the working range of the terminal during use and can unintentionally leave the working range. It is thus important to have methods available that allow such interruptions to be recognized as quickly as possible and allow communications to be resumed in a state that is as well defined as possible.

In order to avoid having every type of error completely interrupt communications and force re-initiation of the entire process, ISO/IEC 14 433-4 defines rules for error detection and correction during the protocol activation phase for Type-A cards. Rather than describe these rules in detail, we refer you to the appropriate section of the standard, where they are presented in an illustrative manner.

Half-duplex protocol in accordance with ISO/IEC 14 433-4

The transmission protocol defined in Part 4 of ISO/IEC 14 433-4 takes into account the special requirements imposed by the use of contactless cards. For instance, it allows a terminal to communicate with several cards in parallel.

Like the $T = 1$ protocol for contact-type cards, this protocol supports a clean separation of layers in accordance with the OSI reference model. Separation of layers means that data belonging to a higher level layer are transmitted completely transparently with respect to lower level layers. This protocol is based on the frames defined for both types of cards (Type A and Type B) in Part 3 of the standard. Four layers are distinguished:

- the physical layer (byte transmission in accordance with 14 433-3)
- the data link layer (for exchanging data blocks in accordance with 14 433-3)
- the session layer (coupled to the data link layer in order to minimize overhead)
- the application layer (where the commands are processed).

This block-oriented protocol starts after the activation sequence has been completed. The terminal is entitled to make the first transmission. This means that after the activation sequence, the card must wait until it receives a block from the terminal. The card responds to each block that it receives by transmitting a response block within a defined frame waiting time. After the response block has been sent, the terminal again holds transmit authorization and the card switches back to reception mode. Communications continue in this manner, with transmit authorization alternately held by the terminal and the card.

The protocol allows several cards to be concurrently activated by a terminal. It can switch back and forth among several cards without having to spend time deactivating a card and activating another card each time it switches cards.

As already mentioned, the probability of errors in data transmissions is higher in systems using contactless cards than in comparable systems using contact-type cards. It is thus especially important for communications between the terminal and the card to take place in the shortest possible time. The data transmission rate, which has a default value of 106 kbit/s, is relatively high compared with the default value of 9.6 kbit/s for the $T = 0$ and $T = 1$ transmission protocols.

Block structure

Each transmission block consists of a leading prolog field, an optional information field and a trailing epilog field. The information field contains data for the application layer.

There are three different types of blocks:

- Information blocks, which are used for the transparent exchange of data belonging to the application layer.
- Reception confirmation blocks (R blocks), which do not contain information fields and serve to indicate positive or negative confirmation of reception.

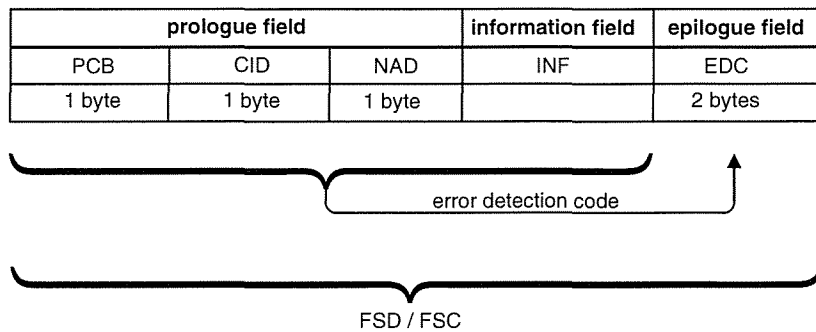


Figure 3.128 Block structure

- System blocks (S blocks), which are used to exchange control data for the protocol. Two types of S blocks are defined: WTX, which is an S block for extending the frame waiting time and has a single-byte information field, and DESELECT, which is an S block with no information field that is used to place the card in the Halt state.

Prologue field The prologue field consists of the protocol control byte (PCB), an optional card identifier (CID) and an optional node address (NAD). The coding of the protocol control byte is shown in Table 3.36.

Table 3.36 Coding of a the protocol control byte (PCB)

Bit	I-block PCB	R-block PCB	S-block PCB	
			DESELECT	WTX
b8	0	1	1	1
b7	0	0	1	1
b6	0 (1 is RFU)	1	0	1
b5	1 = chaining	0 = ACK, 1 = NAK	0	1
b4	1 = CID follows	1 = CID follows	1 = CID follows	
b3	1 = NAD follows	0 (no NAD)	0 (no NAD)	
b2	1	1 (0 is RFU)	1 (0 is RFU)	
b1	block number	block number	0 (1 is RFU)	

Card identifier (CID) The card identifier field is used to identify a particular card, and it also contains information about the power supplied to the card. The most significant two bits (b8 and b7) indicate the power level provided to the card by the terminal. Bits b6 and b5 are set to 0, with 1 being reserved for future use. Bits b4–b1 code the card identifier.

b8	b7	b6	b5	b4	b3	b2	b1
Power level indication		0 (1 = RFU)	0 (1 = RFU)	CID			

Figure 3.129 Format of the card identifier field

The coding of CID is defined in the PPS command for Type-A cards and in the ATTRIB command for Type-B cards. The following rules apply to evaluating the CID field:

- A card that does not support CID ignores all blocks containing a CID.
- A card that supports CID responds to blocks containing its CID by sending back its CID. It ignores blocks that contain other CIDs, and it responds to any block containing a CID of 0 by returning a block with no CID.

These rules allow the terminal to communicate concurrently with several active cards without having to deactivate cards that are not being addressed.

Table 3.37 Coding of the power level indication

b8	b7	Meaning
0	0	The card does not support power level indication
0	1	The amount of power is insufficient for full functionality
1	0	Adequate power for full functionality
1	1	More than adequate power for full functionality

Node address (NAD) The third byte of the prologue field is called the ‘node address’. The node address is used to establish and address certain types of logical connections between the card and the terminal. Node addresses are used in the same manner as for contact-type cards. They are defined in ISO/IEC 7816-3 and described in Chapter 6.

Information field (INF) In I blocks, the information field acts as a container for data for the application layer. The content of this field is transferred fully transparently. In S blocks, the information field is used to control extending the frame waiting time.

Frame waiting time (FWT)

In order to achieve a defined termination of communications with a non-responding card in the shortest possible time, a ‘frame waiting time’ (FWT) is defined. It corresponds to the block waiting time of the T = 1 protocol for contact-type cards. The frame waiting time is the maximum interval between the end of a frame transmitted by the terminal and the start of the response frame from the card. If this interval expires without a response from the card, the terminal assumes that there is a malfunction in the card and reacquires transmit authorization in order to initiate error-detection mechanisms. As previously described, the ATS for Type-A cards contains the value of the frame waiting time integer (FWI) in the TB1 interface byte. The frame waiting time can be calculated from the FWI using the following formula:

$$\text{FWT} = (256 \times 16 \div f_C) \times 2^{\text{FWI}}$$

For Type-B cards, the FWI is defined in the ATQB (Answer to Request, Type B). The minimum value of FWT, which is obtained when the value of FWI is 0, is approximately 302 μs . The maximum value of FWT (FWT_{MAX}), which is obtained when the value of FWI is 14, is

approximately 4949 ms. The normal command processing time of the card must be taken into account when selecting the frame waiting time. If the value of the frame waiting time is set too large, it will take longer for the terminal to detect that a card is not responding. In practice, cards can respond relatively quickly to most commands, but a few commands, such as commands involving the computation of a cryptographic algorithm, require significantly more time to execute. In order to avoid having to use a long frame waiting time for all commands just to accommodate such cases, there is a mechanism for extending the waiting time. This allows the card to request an extension of the frame waiting time for an individual command.

Extending the frame waiting time

To request an extension of the FWT, the card transmits a special S block called ‘S(STX) request’. It receives a corresponding ‘S(WTX) response’ from the terminal to confirm the request. The terminal is not allowed to deny such a request.

The length of the extension to the frame waiting time is sent to the terminal using one byte in the information field of the S(WTX) S block. The new, temporary frame waiting time for processing the current command is obtained by multiplying this value by the frame waiting time.

b8	b7	b6	b5	b4	b3	b2	b1
Power level indication		WXTM (1–59; 0 and 60–63 are RFU)					

Figure 3.130 Format of the INF field of an S(WTX) request

b8	b7	b6	b5	b4	b3	b2	b1
		WXTM					

Figure 3.131 Format of the INF field of an S(WTX) response

The temporary frame waiting time (FWT_{TEMP}) is measured starting with the end of the S(WTX) response sent by the terminal. It is calculated using the following formula:

$$\text{FWT}_{\text{TEMP}} = \text{FWT} \times \text{WXTM}$$

If the formula yields a result greater than FWT_{MAX} (≈ 4949 ms), the value of FWT_{MAX} must be used instead of the calculated value of FWT_{TEMP} .

Block chaining

The chaining function allows either one of the communicating parties to transmit data blocks that are too big to fit within a single frame by partitioning the data into several I frames sent in succession. Each of these chained I blocks has a length that is less than or equal to the frame length specified by FSC or FSD, as appropriate.

When block chaining is used, the sender sets the chaining bit in the protocol control byte (PCD) of the first block of the chain. This indicates to the recipient that the block chaining

function is being used and that the subsequent block contains chained data. If the recipient receives the first block of the chain correctly, it indicates correct reception and its readiness to receive the next block by returning an R block with the same block number as the block it just received. The next block can then be sent. This back-and-forth exchange of I blocks and R blocks continues until the sender transmits an I block in which the chaining bit is not set. On receiving this block, the recipient knows that all of the application-layer data have been received, so it can process this data block and send the associated response.

Deactivating a card

When data transmission between the terminal and the card is completely finished, the terminal places the card in the Halt state by sending it a DESELECT command, which is transmitted using an S block. The card responds to this command with an S(DESELECT) response block and enters the Halt state.

Error handling

The block transmission protocol has error-detection mechanisms that are similar to those of the T = 1 protocol and allow resynchronization at various levels in case of transmission errors. The exact rules for the protocol processes can be found in Part 4 of ISO/IEC 14 433. Extensive examples of error-free protocol processes and error handling can also be found in the Annex to the standard.

3.6.4 Vicinity integrated circuits cards (ISO/IEC 15 693)

The ISO/IEC 15 693 standard, whose exact title is ‘Identification cards – Contactless integrated circuit(s) cards - Vicinity cards’, describes the properties and operating modes of contactless smart cards having a range up to 1 m. This type of card is preferred for applications such as access control, since a range of around 1 m means that it is not necessary for the card to be held in the user’s hand. Instead, it can remain in the user’s pocket, purse or other location. Up to now, this standard has not found widespread use in smart card systems, so we omit providing an extensive description of it here.

3.6.5 Test methods for contactless smart cards

The ISO/IEC 10 373 standard contains a compilation of all test methods for ID cards with and without chips. ISO/IEC 10 373 consists of seven parts, as shown below. As can be seen from this following list, three parts of this standard contain special test methods for contactless cards.

- Part 1: General characteristics tests
- Part 2: Cards with magnetic stripes
- Part 3: Integrated circuit(s) cards with contacts and related devices
- Part 4: Close-coupled cards

- Part 5: Optical memory cards
- Part 6: Proximity cards
- Part 7: Vicinity cards.

3.6.5.1 Part 4: Test methods for close-coupling smart cards

This part of the standard describes methods for testing the physical interfaces of contactless close-coupling smart cards compliant with ISO/IEC 10 536. Test aids in the form of reference coils and capacitive coupling surfaces are defined for use in measuring energy and data transfers between the terminal and the smart card and verifying conformance with the values specified in the standard (ISO/IEC 10 536).

3.6.5.2 Part 6: Test methods for proximity-coupling smart cards

Part 6 of the standard describes methods for testing the physical interfaces of contactless proximity-coupling smart cards compliant with ISO/IEC 14 433. The test aids necessary for this purpose, which consist of a calibration coil, a test setup for measuring load modulation and a reference card, are defined in the standard. Test methods for the following properties of the card or terminal are described in the standard:

- the resistance of the card to damage by electrostatic discharge
- the amplitude of the load modulation and the functionality of the card within its defined modulation region, as described in the basic standard
- the strength of the field generated by the terminal
- the modulation index and transient behavior (rise and fall times, overshoots etc.) of the signal generated by the terminal.

It must be noted that the small amplitude of the load modulation signal means it is difficult to make accurate and reproducible measurements of this signal, and it is to be hoped that suitable measurement equipment will soon be commercially available. Until such time, it is advisable to request assistance from suppliers of cards and/or terminals and agree on terms and conditions of delivery at an early date.

3.6.5.3 Part 7: Test methods for vicinity-coupling smart cards

Part 7 of the standard describes methods for testing the physical interfaces of vicinity-coupling smart cards compliant with ISO/IEC 15 693. The test aids and methods largely correspond to those in Part 6 of the standard. The only difference is in the construction of the reference card, due to the different subcarrier frequency.

4

Informatic Foundations

‘A smart card is a small computer in credit-card format with no man–machine interface’. This statement expresses an essential fact, which is that in contrast to all other types of cards, the specific properties of smart cards are determined by the microcontroller integrated into the card.

The primary function of the plastic body of the card is to carry the microcontroller. Of course, other components may be present in addition to the microcontroller, but they are not essential to the actual smart card functions. A basic understanding of certain aspects of informatics is necessary to understand the characteristics of these small computers and the IT mechanisms based on them.

Our intention here is not to convey expert knowledge. That is anyhow not necessary for understanding the basic features of the procedures and techniques used with smart cards. The basic knowledge that this chapter offers provides a fully adequate level of comprehension. Consequently, the information presented here delves into the technical details only to the extent necessary to understand fundamental relationships.

Nearly half of this chapter is dedicated to cryptographic procedures used in the field of smart cards. Until a few years ago, the subject of cryptography was surrounded by a veil of secrecy and ignorance. However, this situation has changed dramatically in recent years, and there is now an extensive literature on this subject. As with the sections of this chapter that deal with the general aspects of information technology, here we provide only the basic information necessary for understanding cryptographic algorithms and protocols. For more detailed information, we refer you to the well-known books on this subject, such as those by Bruce Schneier [Schneier 96] and Alfred Menezes [Menezes 97]. A further rich source of information on cryptography is the World Wide Web, where you will find the home pages of several research institutes (such as [GMD]), standards organizations (such as [ETSE, IEC, ISO]), agencies (such as [BSI, NSA]), companies (such as [Certicom, Counterpane, R3, RSA]), associations (such as [CCC, Teletrust]) and individuals with an interest in the subject (such as [Gutmann]).

4.1 STRUCTURING DATA

Storing or transmitting data unavoidably requires an exact definition of the data in question and their structure. Only then is it possible to subsequently recognize and interpret the data elements. Fixed-length data structures with non-modifiable sequences regularly cause systems to ‘collapse’. The best example of this is the conversion of the many different European currencies to the euro. All systems and data structures with fixed currency definitions had to be upgraded at considerable cost. The same difficulties manifest themselves in many smart card applications. Fixed data structures that need to be extended or shortened sooner or later give rise to considerable effort and expense.

However, the problem of structuring data has been around for a long time, and there is an adequate choice of methods that can be used to solve the problem. One method that is very popular in the world of smart cards, and which is coming into more general use in informatics, comes from the field of data transmission. It is called Abstract Syntax Notation 1, or ASN.1 for short. This is a coding-independent description of data objects, originally developed for transmitting data between different computer systems. An alternative to ASN.1 would be using extensible markup language (XML) to structure data, but up to now this method has not gained a foothold in real applications in the smart card world.

In principle, ASN.1 is a sort of artificial language that is suitable for describing data and data structures, rather than programs. The syntax is standardized in ISO/IEC 8824, and the coding rules are defined by ISO/IEC 8825. Both of these standards were developed from Recommendation X.409 of the CCITT.

Describing ASN.1 in detail would require a book on its own, so here we only address a few essential aspects in order to give a general ideal of how it works. For further information, we suggest you consult the relevant literature, such as Walter Gora [Gora 98].

ASN.1 has a number of elementary (‘primitive’) data types and composite (‘constructed’) data types. It is also possible to extend the syntax of ASN.1 using macros in order to obtain any desired enhancements to ASN.1. Listings 4.1 through 4.3 show some simple examples of how ASN.1 can be used, including defining and coding data.

Table 4.1 Some of the data types used in ASN.1

Data type	Sort	Meaning
BOOLEAN	Primitive	Boolean value: yes/no
INTEGER	Primitive	Negative and positive integers
OCTET STRING	Primitive	Byte sequence (one byte = one 8-bit octet)
BIT STRING	Primitive	Bit sequence
SEQUENCE	Constructed	Several components combined to form a new data type

The basic idea of coding data using ASN.1 is to prefix each data object with a unique label and information about its length. The rather complex syntax of the description language also allows users to define their own data types and nest data objects. The original idea, which was to create a generally valid syntax that could form the basis for data exchange between fundamentally different computer systems, is scarcely used in smart cards. Currently, only a very small part of the available syntax is used in this area, mainly due to the very limited memory capacity of smart cards.

Listing 4.1 A simple example of data type definition using ASN.1

SC_Controller ::= SEQUENCE { Name IA5String, CPUType CPUPower, NPU BOOLEAN, EEPROMSize INTEGER, RAMSize INTEGER, ROMSize INTEGER}	Definition of a new data type for SC_Controller. The name of the microcontroller is an ASCII string. CPUType refers to the definition of CPUPower. Boolean value as a yes/no assertion regarding whether a coprocessor (NPU) is present. The size of the EEPROM is an integer value. The size of the RAM is an integer value. The size of the ROM is an integer value.
CPUPower ::= ENUMERATED { 8Bit (8), 16Bit (16), 32Bit (32)}	Definition of a new data type for CPUPower as an enumerated type. Possible selection value for the 8-bit CPU type. Possible selection value for the 16-bit CPU type. Possible selection value for the 32-bit CPU type.

Listing 4.2 The data definitions from Listing 4.1, filled with data for a particular microcontroller

SuperXS SC_Controller ::= { Name "XS 8 Bit", CPUType 8, NPU true, EEPROMSize 1024, RAMSize 256, ROMSize 8192}	Specific instance of the SC_Controller data type with the data for SuperXS. The name of the microcontroller is 'XS 8 Bit'. This is an 8-bit CPU. No coprocessor (NPU) is present. The size of the EEPROM is 1024 bytes. The size of the RAM is 256 bytes. The size of the ROM is 8192 bytes.
---	--

Listing 4.3 The data for a particular microcontroller from Listing 4.2, coded using the ASN.1 BER

'30 1C'	Tag '30' for a string with a length of 28 bytes ('1C').
'16 08 58 53 20 38 20 42 69 74'	Tag '16' for an IA5 string with a length of 8 bytes ('08') and a content of '58 53 20 38 20 42 69 74' (= "XS 8 Bit").
'0A 01 08'	Tag '0A' for an enumerated data type with a length of 1 byte ('01') and a content of '08'.
'01 01 FF'	Tag '01' for a Boolean data type with a length of 1 byte ('01') and a content of 'FF', which corresponds to the value 'true'.
'02 02 04 00'	Tag '02' for an integer data type with a length of 2 bytes ('02') and a content of '04 00' (1024).
'02 02 01 00'	Tag '02' for an integer data type with a length of 2 bytes ('02') and a content of '01 00' (256).
'02 02 20 00'	Tag '02' for an integer data type with a length of 2 bytes ('02') and a content of '02 00' (8192).

The Basic Encoding Rules (BER) for ASN.1 are defined in the ISO/IEC 8825 standard. Data objects created according to these rules are called BER-TLV-coded data objects. A BER-coded data object has a label (called a 'tag'), a length field and the actual data part, with an optional end marker. Certain bits in the tag are predefined by the coding rules. The actual structure is shown in Figure 4.1. The Distinguished Encoding Rules (DER) form a subset of the BER. These coding rules specify, among other things, the coding of the length information, which may be one, two or three bytes long. A basic summary of the BER and DER can be found in Burton Kaliski [Kaliski 93].

ASN.1 objects are coded using the classic TLV structure, in which 'T' (tag) denotes the object's label, 'L' (length) refers to its length and 'V' (value) is the actual data. The first field of a TLV structure is the tag for the data object in the following V field. To avoid the need for each user to define his or her own tags, which would open the door to incompatibility, there are standards that define tags for various, frequently used data structures. ISO/IEC 7816-6, for example, defines tags for objects used in general industrial applications, ISO/IEC 7816-4 defines tags for secure messaging, and EMV also defines several other tags. It is by no means the case that a given tag is universally used for the same type of data element, but a process of standardization is essentially taking place.

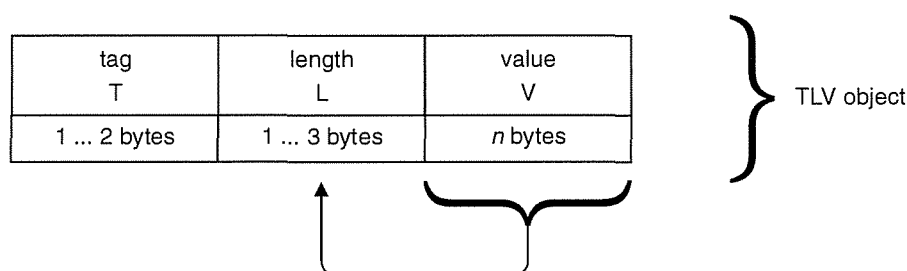


Figure 4.1 The principle of BER-based TLV coding according to ANS.1

The two most significant bits of the tag encode the class of the following data object. The class indicates the general type of the data object. The *universal* class indicates general data objects, such as integers and character strings. The *application* class indicates that the data object belongs to a particular application or standard (e.g. ISO/IEC 7816-6). The other two classes, *context-specific* and *private*, fall under the heading of non-standardized applications.

The bit following the two class bits indicates whether the tagged object is constructed from other data objects. The five least-significant bits are the actual label. Since this can have a value of only 0 through 30, due to its limited address space, it is possible to point to the following byte by setting all five bits to 1. All values from 31 to 127 are allowed in the second byte. Bit 8 of the second byte is a pointer that is reserved for future use, so it cannot presently be used. The required number of length bytes is shown in Table 4.3.

The standard also defines the term 'template'. A template is a data object that serves as a container for other data objects. ISO/IEC 7816-6 defines the tags for possible data objects in the domain of industry-wide applications of smart cards. ISO 9992-2 covers the domain of smart card financial transactions.

This method of data encoding has several characteristics that are particularly useful in the field of smart cards. Since the available memory space is generally never enough, using data objects based on ASN.1 can produce considerable space savings. TLV encoding makes it possible to transfer and store variable-length data without a lot of complications. This allows

Table 4.2 ASN.1 tag coding

Byte 1	b8	b7	b6	b5	b4	b3	b2	b1	Meaning
	0	0	...	...	...	...	...	...	Universal class
	0	1	...	...	...	...	...	...	Application class
	1	0	...	...	...	...	...	...	Context-specific class
	1	1	...	...	...	...	...	...	Private class
	...	...	0	...	...	...	...	...	Primitive data object
	...	...	1	...	...	...	...	...	Constructed data object
	...	...	...	X	X	X	X	X	Tag code (0–30)
	...	...	...	1	1	1	1	1	Pointer to the following byte (byte 2), which specifies the tag code
Byte 2	b8	b7–b1							Meaning
	0	31–127							Tag code

Table 4.3 Structure of the DER length field in ASN.1

Byte 1	Byte 2	Byte 3	Meaning
0-127	—	—	One byte is needed for these length values
'81'	128-255		Two bytes are needed for these length values
'82'	256-65,535		Three bytes are needed for these length values

memory to be used very economically. This is illustrated in Figure 4.2, which shows the TLV encoding of a name.

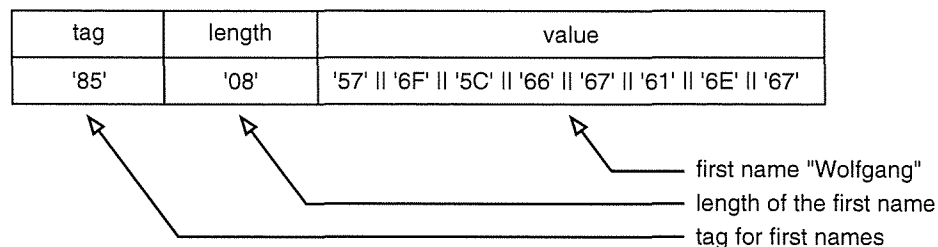


Figure 4.2 TLV encoding of the name ‘Wolfgang’

Subsequent extensions to data structures can be undertaken very easily with ASN.1, since all that is necessary is to insert additional TLV-coded data objects into the existing data structure. Full compatibility with the previous version is retained as long as the previous TLV objects are not deleted. The same is true of new versions of data structures in which changes have been made with respect to the previous coding. This is a straightforward process that only requires modifications to the tags. It is equally simple to represent the same data using different codings. Collectively, these advantages explain why the ASN.1 syntax, based on TLV coding, is particularly popular in the smart card industry.

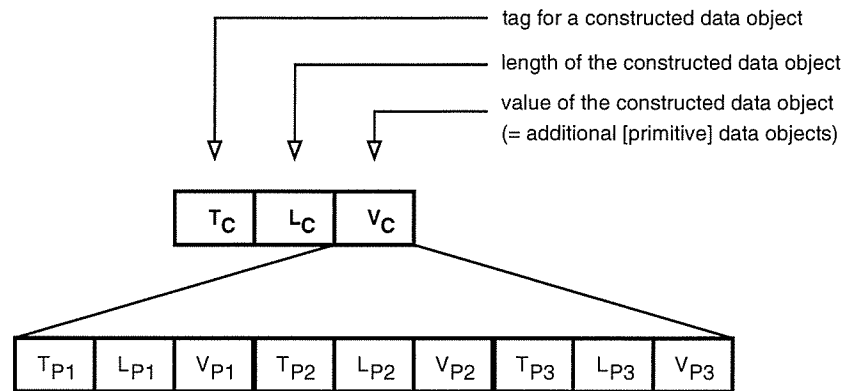


Figure 4.3 Basic scheme for forming constructed TLV-coded data structures from several primitive TLV-coded data objects. The indices 'C' and 'P' stand for 'constructed' and 'primitive'

The main disadvantage of ASN.1 data objects is that the administrative data overhead is rather high if the volume of user data is small. For example, if the user data is only one byte, two additional bytes (tag and length) are still needed for its administrative data. However, the larger the volume of the user data, the more favorable is the relationship. The ASN.1 structured data in the German health insurance card form a good example of this. There are between 70 and 212 bytes of user data. The administrative data amount to 36 bytes, which means that the administrative overhead ranges from 17 to 51 %.

We can recapitulate all the above with a further example. Suppose we wish to store surnames, given names and titles in a file with a transparent data structure. Irrespective of the proper ASN.1 description, the TLV-coded data will have the structure shown in Figure 4.4. The tags used in this example have been freely chosen and thus do not correspond to any relevant standard.

version 1	T	L	V	T	L	V	T	L	V
	'85'	'07'	"Manfred"	'87'	'05'	"Meier"	'84'	'04'	"Ing."
version 2	T	L	V	T	L	V	T	L	V
	'84'	'04'	"Ing."	'85'	'07'	"Manfred"	'87'	'05'	"Meier"
version 3	T	L	V	T	L	V	T	L	V
	'87'	'05'	"Meier"	'85'	'07'	"Manfred"	'84'	'04'	"Ing."

Figure 4.4 An example of sequence independence within a TLV structure

When evaluating this data structure, the computer compares the first tag with all tags known to it. If it finds a match, then it recognizes the first object as a given name. It reads the length of this object from the next byte. The subsequent bytes are then the actual object, i.e. the given

name. This is followed by the next TLV object, whose first byte is the tag for a surname. The computer recognizes this using exactly the same process as for the first object.

If it becomes necessary to extend the data structure, e.g. by adding a title, a new type of data object can simply be inserted into the existing structure. The insertion point is unimportant. The extended structure remains fully compatible with the previous version, since the new type of data object receives its own tag and is thus unambiguously identified. Programs that only know the old tags will not be upset by the new one, since they do not recognize it and thus automatically skip it. Other programs that do know the new tag can evaluate it, but even if the old structure is used, they will not experience any problems.

4.2 CODING ALPHANUMERIC DATA

The alphanumeric data in the files and data objects of smart cards can be stored in a wide variety of formats. In part, this is a result of intensive memory space optimization measures and a lack of general agreement among the various applications and specifications, and in part it is due to the triumphant progress of smart cards in countries outside of Western Europe, which have their own alphabets. In such situations, the original 7- and 8-bit character sets must be replaced by more powerful coding schemes for alphanumeric data.

4.2.1 7-bit code

A total of 128 (2^7) characters can be represented using a 7-bit code. The most widely used international 7-bit code, which is commonly known as the ASCII (American Standard Code for Information Interchange) code, is specified in ISO/IEC 646. The importance of ASCII has been steadily decreasing for many years, since the number of characters it can represent is much too small.

4.2.2 8-bit code

The most commonly used 8-bit code ($2^8 = 256$ characters) is derived from the 7-bit ASCII code and is standardized in ISO/IEC 8859. It consists of two 7-bit code tables specifying control characters and printable characters. The lower order table is identical to the 7-bit ASCII table and is always the same. The higher order table can vary to accommodate a country-specific character set. Probably the best-known higher-order code table is Latin 1, which contains the characters specific to the countries of Western Europe. Latin 2, by contrast, contains the special characters for the East European countries. ISO/IEC 8859 consists of 16 parts in total, which define a series of higher order code tables for the character sets of various languages.

The characters of the Latin 1 code table in ISO/IEC 8859 are also found in a slightly modified coding in DOS as 'Code Table 850' according to the IBM register, in the form of 'PC ASCII' and as 'ANSI code' under Windows.

EBCDIC ('extended binary coded decimal interchange code'), which is widely used in mainframe computers, is not used with smart cards.

Table 4.4 The 7-bit standard character table specified for the GSM system by GSM 03.38, which is based on the ASCII character set. Each character is shown at the top center of each cell, with the 7-bit code in decimal notation at the bottom left and hexadecimal notation at the bottom right. The following abbreviations are used: CR = carriage return, LF = line feed, SP = space

@	Δ	SP	0	i	P	ı	p
0 '00'	16 '10'	32 '20'	48 '30'	64 '40'	80 '50'	96 '60'	112 '70'
£	—	!	ı	A	Q	a	q
1 '01'	17 '11'	33 '21'	49 '31'	65 '41'	81 '51'	97 '61'	113 '71'
\$	Φ	"	2	B	R	b	r
2 '02'	18 '12'	34 '22'	50 '32'	66 '42'	82 '52'	98 '62'	114 '72'
¥	Γ	#	3	C	S	c	s
3 '03'	19 '13'	35 '23'	51 '33'	67 '43'	83 '53'	99 '63'	115 '73'
è	Λ	α	4	D	T	d	t
4 '04'	20 '14'	36 '24'	52 '34'	68 '44'	84 '54'	100 '64'	116 '74'
é	Ω	%	5	E	U	e	u
5 '05'	21 '15'	37 '25'	53 '35'	69 '45'	85 '55'	101 '65'	117 '75'
ù	Π	&	6	F	V	f	v
6 '06'	22 '16'	38 '26'	54 '36'	70 '46'	86 '56'	102 '66'	118 '76'
ì	Ψ	,	7	G	W	g	w
7 '07'	23 '17'	39 '27'	55 '37'	71 '47'	87 '57'	103 '67'	119 '77'
ò	Σ	(	8	H	X	h	x
8 '08'	24 '18'	40 '28'	56 '38'	72 '48'	88 '58'	104 '68'	120 '78'
Ç	Θ	)	9	I	Y	i	y
9 '09'	25 '19'	41 '29'	57 '39'	73 '49'	89 '59'	105 '69'	121 '79'
LF	Ξ	*	:	J	Z	j	z
10 '0A'	26 '1A'	42 '2A'	58 '3A'	74 '4A'	90 '5A'	106 '6A'	122 '7A'

Table 4.4 (Cont.)

Ø	1)	+	;	K	Ä	k	ä
11 '0B'	27 '1B'	43 '2B'	59 '3B'	75 '4B'	91 '5B'	107 '6B'	123 '7B'
ø	Æ	,	<	L	Ö	l	ö
12 '0C'	28 '1C'	44 '2C'	60 '3C'	76 '4C'	92 '5C'	108 '6C'	124 '7C'
CR	æ	-	=	M	Ñ	m	ñ
13 '0D'	29 '1D'	45 '2D'	61 '3D'	77 '4D'	93 '5D'	109 '6D'	125 '7D'
Å	ß	.	>	N	Ü	n	ü
14 '0E'	30 '1E'	46 '2E'	62 '3E'	78 '4E'	94 '5E'	110 '6E'	126 '7E'
å	É	/	?	O	§	o	à
15 '0F'	31 '1F'	47 '2F'	63 '3F'	79 '4F'	95 '5F'	111 '6F'	127 '7F'

4.2.3 16-bit code (Unicode)

Codes with a width of 16 bits allow 65,536 (2^{16}) characters to be represented. The only example of such a code is Unicode, which was developed as a private initiative by the Unicode Consortium [Unicode] as an industry standard.

The first 256 Unicode characters are identical to ISO/IEC 8859 Latin 1, so there is at least upward compatibility in this part of the character coding. Although the number of characters that can be represented with a 16-bit code is sufficient to represent the characters of the most important living languages, it is unfortunately not sufficient to represent all existing characters.

To compensate for this, a sort of escape sequence ('surrogate pairs') has been incorporated into the 16-bit character code in the current version of Unicode (3.0). This allows a supplementary byte to be used, so that up to one million characters can be represented.

4.2.4 32-bit code (UCS)

Unicode was originally limited to 65,536 characters. Although this limitation does not cause problems in everyday use, it can be avoided by using an extended character coding scheme. ISO/IEC 10646 specifies a 32-bit code called the 'Universal Character Set' (UCS), which allows 4,294,967,296 (2^{32}) characters to be represented, although only half of the available codes (2,147,483,648) are actually used.

The four bytes of the UCS are called (in decreasing order of significance) *group*, *level*, *row* and *cell*. UCS thus consists of 256 groups of 256 levels, each of which has 256 rows of 256 cells. A level thus specifies 65,536 characters. The lowest level, which is Group 0, Level 0, is called the 'basic multilingual plane' (BMP) and is identical to Unicode. The lowest row, which

is Group 0, Level 0, Row 0, thus automatically corresponds to the character set of ISO/IEC 8859 Latin 1, and the first 128 characters are thus identical to the ASCII code.

This can be illustrated using a brief example. The letter "A" is coded as '30' in 7-bit ASCII and 8-bit ISO/IEC 8859 Latin 1. Since the first 256 characters of Unicode are identical to ISO/IEC 8859 Latin 1, the letter "A" is coded as '00 30' in 16-bit Unicode and as '00 00 00 30' in 32-bit UCS. UCS is the only character coding scheme that allows all characters of all living and dead languages to be coded using unique numerical values. Consequently, UCS is the most important coding scheme for future use, despite its memory requirement of four bytes per character.

There are three commonly used schemes, called 'UCS transition formats' (UTFs), for translating the codes of 32-bit UCS and 16-bit Unicode characters. UTF-8 translates characters into variable-length byte strings whose least-significant seven bits correspond to ASCII. UTF-16 uses 16 bits for coding and thus corresponds to the BMP of UCS, which also uses two bytes for coding. UTF-16 is also referred to as 'UTF-2', since it uses two bytes for coding. UTF-32 corresponds to the usual four-byte representation of UCS, for which reason it is also referred to as 'UCS-4'.

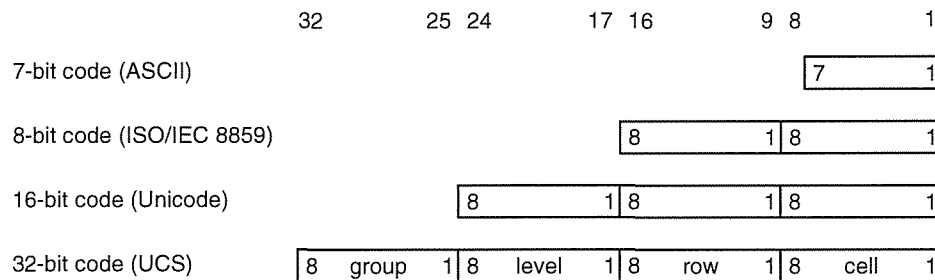


Figure 4.5 The relationships between the internationally most commonly used 7-, 8-, 16- and 32-bit codes for alphanumeric characters

4.3 SDL NOTATION

This book uses SDL notation to describe states and state transitions. For some years, this approach has been used ever more frequently in the smart card domain to describe state-oriented mechanisms, such as those used for communication protocols. 'SDL' stands for 'Specification and Description Language', and it is described in detail in CCITT Recommendation Z.100.

SDL notation is similar to the notation used in standard flowcharts. However, it does not describe program flows, but instead states and state transitions. SDL diagrams are constructed using standardized individual symbols interconnected by lines. The flow is always from top left to bottom right, so the lines connecting individual symbols do not need arrowheads to identify their start and end points.¹

In simplified form, the notation can be regarded as a description of a system consisting of a certain number of processes, where each process is a state machine. If a state machine is in a stable state, it can receive a signal from outside. Depending on the data it receives, the

¹ For a detailed example of an SDL diagram, see Section 6.4.2, 'The T = 0 transmission protocol'

machine may then attain a specific new state. Additional actions may occur between the initial and final states, such as receiving and transmitting data or computing a value.

Figure 4.6 shows the 10 symbols used in this book. They are a selection from a much larger set defined in Z.100, but they suffice as a basic set for use with smart cards. The Start symbol (1) denotes the beginning of a process. Most SDL diagrams begin with this symbol. The Task symbol (2) indicates a specific activity, which is described by text within the box. With this symbol, there is no additional detailed description in the form of a subroutine. The Decision symbol (3) allows a query during a state transition, to which the answer may be 'yes' or 'no'. The Label symbol (4) marks a link to another SDL diagram and is primarily used to divide large diagrams into several smaller diagrams.

The Input (5) and Output (6) symbols represent interfaces to the outside world. The exact input and output parameters are described inside the symbol. The State symbol (7) is used to describe a state. The state attained at each stage is indicated by this symbol.

The final three symbols describe subroutines. The Subroutine symbol (8) indicates that the content of this box is described in more detail elsewhere. The Subroutine start (9) and Subroutine end (10) symbols delimit the detailed description of a subroutine.

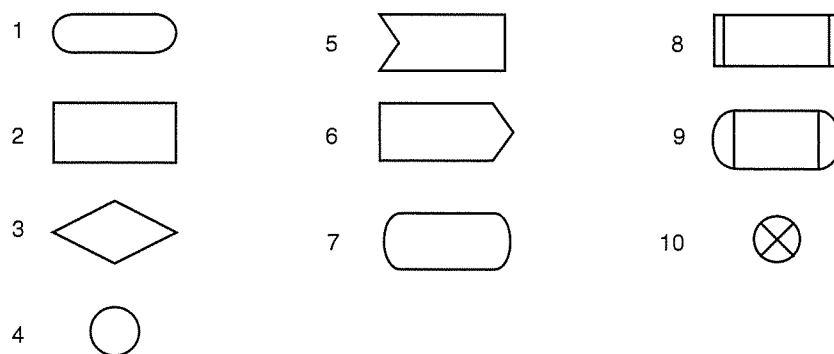


Figure 4.6 The SDL notation symbols used in this book, which are compliant with CCITT Z.100:

1 – Start	5 – Input	8 – Subroutine
2 – Task	6 – Output	9 – Subroutine start
3 – Decision	7 – State	10 – Subroutine end
4 – Label		

4.4 STATE MACHINES

A state machine is a type of automaton. A common example of an automaton is a vending machine, into which you insert a coin and then press a button. After this, you can open a compartment and remove your selection. In slightly more abstract terms, this automaton defines a chain of events involving various state transitions. In the initial state, the automaton waits for money to be inserted. Any other action, such as pressing a button, will not cause anything to happen. Only the insertion of a coin causes a transition from the initial state to the 'money inserted' state. The next transition occurs as a result of pressing the button, following which the automaton allows a compartment to be opened.

In informatics, state machines can be very effectively visualized using graphs or Petri networks. These are not only useful for modeling state machines; they can also be used to investigate certain properties of the systems they describe. The objectives are to identify any deadlocks that may occur in the process and ensure correct command processing.

4.4.1 Basic theory of state machines

The objective of this section is to provide an introduction to state diagrams, which are used to describe smart card applications, and a general explanation of how to interpret them.

A state diagram is a type of graph that represents a set of states and the interrelationships of these states. The states are shown as nodes, and their relationships are shown as lines. If a line indicates a direction, which means that it has an arrowhead at one end, it is called a 'directed line' and the graph is a 'directed graph'. The arrow indicates the direction in which a state transition can take place. The actual placement of the nodes and lines in the graph plays no part in the interpretation of the diagram. A sequence of nodes connected by lines is called a path. If the first and last nodes are the same and there is more than one node, the path is called a loop.

This is only a very small part of graph theory, but it is essentially all we need to be able to describe states and their associated state machines in smart card applications.

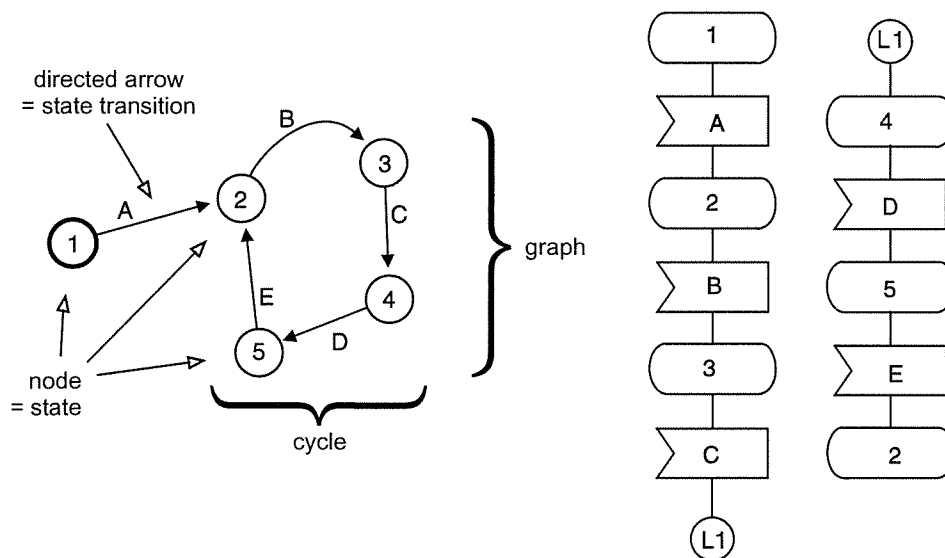


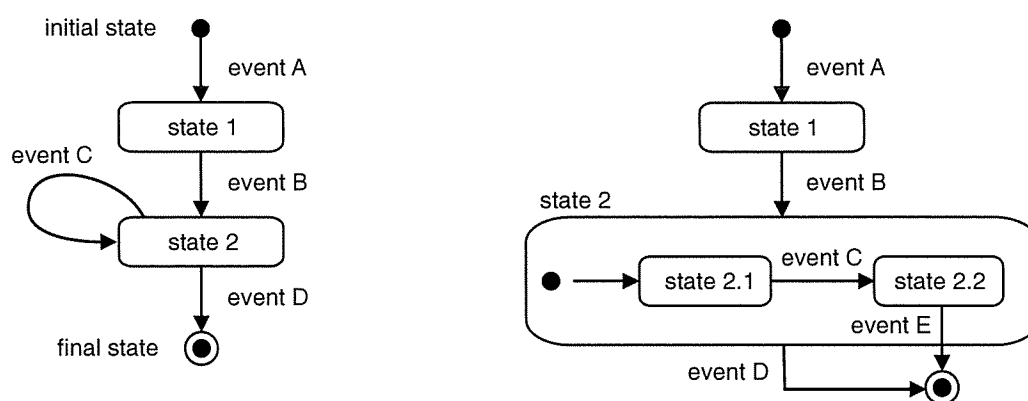
Figure 4.7 Examples of two different representations of state diagrams. On the left is a directed state diagram, and on the right an equivalent SDL diagram

4.4.2 Practical applications

An additional advantage of microprocessor cards compared with simple memory cards is that the command sequences can be specified in advance. It is thus possible to precisely specify all commands in terms of their parameters and sequence. In combination with object-oriented access authorization for files, this provides additional protection against unauthorized access.

Table 4.5 Description of the state diagram of Figure 4.6 in tabular form

State after transition	State before transition				
	1	2	3	4	5
1	—	—	—	—	—
2	A	—	—	—	E
3	—	B	—	—	—
4	—	—	C	—	—
5	—	—	—	D	—

**Figure 4.8** Two sample state diagrams using UML notation. The diagram on the left shows a simple sequential state flow, while the diagram on the right shows a state diagram with substates

However, the possibilities offered by smart cards in this respect vary greatly. Simple operating systems usually cannot manage state machines, while with modern operating systems it is even possible to define application-specific state machines that work with command parameters.

A typical example of a simple state machine is provided by the two commands needed to authenticate a terminal. The first command asks the card for a random number. This activates a state machine that accepts only an authentication command as the next command. If the card receives this command, the process completes and all other types of command are allowed. If the card receives any command other than an authentication command, the state machine generates an error message and the process is aborted. The command sequence must then be restarted from the beginning.

Such simple state machines have several major advantages in smart cards. Since they are limited to very few commands in a rigidly defined sequence, they require little memory space and program overhead. In many applications, it is sufficient to protect file contents using object-oriented access mechanisms, without imposing any other restrictions on command sequences. Only a few procedures, such as authentication, must follow prescribed sequences. This can be implemented with very little memory using simple state machines.

These simple state machines can be extended to verify all commands, along with all of their parameters, within a defined graph before they are executed. Depending on how the state machine is constructed, under certain conditions it may be possible to dispense with

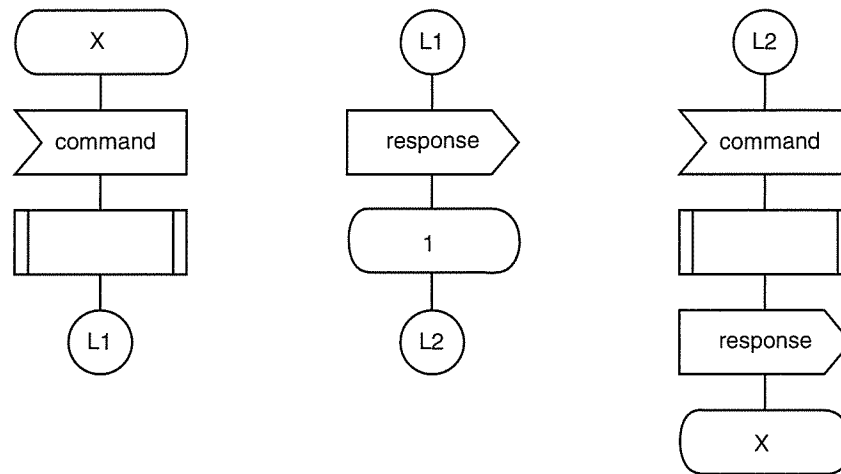


Figure 4.9 An example of a simple smart card state machine with two states, ‘X’ and ‘1’

object-oriented file access protection, since the state machine can perform all the necessary checks before a command is actually executed. Of course, an error in the state diagram could have fatal consequences for the security of the system. As it is very difficult to verify the complete absence of errors in the state diagrams of complex state machines, file access protection is still used in practice. Correctly describing all of the processes and commands present on a smart card is very time-consuming, so it is often necessary to do this empirically to a certain extent.

Now that we have described the advantages of state machines, we must mention their drawbacks. Implementing a state machine with the required capabilities is very time-consuming in terms of both design and subsequent programming. Since a state machine is controlled by the stored representation of a graph, a considerable amount of program memory is needed just to hold a state machine, since the graph must be stored in memory in addition to the actual state machine. The amount of memory space naturally depends on the complexity of the graph to be executed. The amount of information contained in a graph having many states and a corresponding number of transitions can be very large relative to typical smart card memory capacities.

State machines for smart cards are addressed by the ISO/IEC 7816-9 standard. It describes ‘access control descriptors’ (ACDs), which define the commands that are permitted in a specific state, along with their associated parameters. A smart card operating system can monitor hard-coded state machines using these ACDs.

In order to illustrate the capabilities of a state machine in summary form, Figure 4.10 shows the state diagram for a small application. Its operation is described below.

After a reset, the smart card is in the initial state, denoted by 1. In this state, every file in the directory may be selected using SELECT FILE; this does not cause a state transition. All other commands except PIN verification (VERIFY) are prohibited, and the card responds to such commands with an error message. After successful verification of the PIN, the state machine changes to state 2.

Two commands are permitted in state 2. The first path leads via SELECT FILE to state 3, where the selected file may be read. The second path originating from state 2 leads to state 4 after the terminal requests a random number from the card (ASK RANDOM). From here, any command other than EXTERNAL AUTHENTICATE leads back to the initial state (1). When

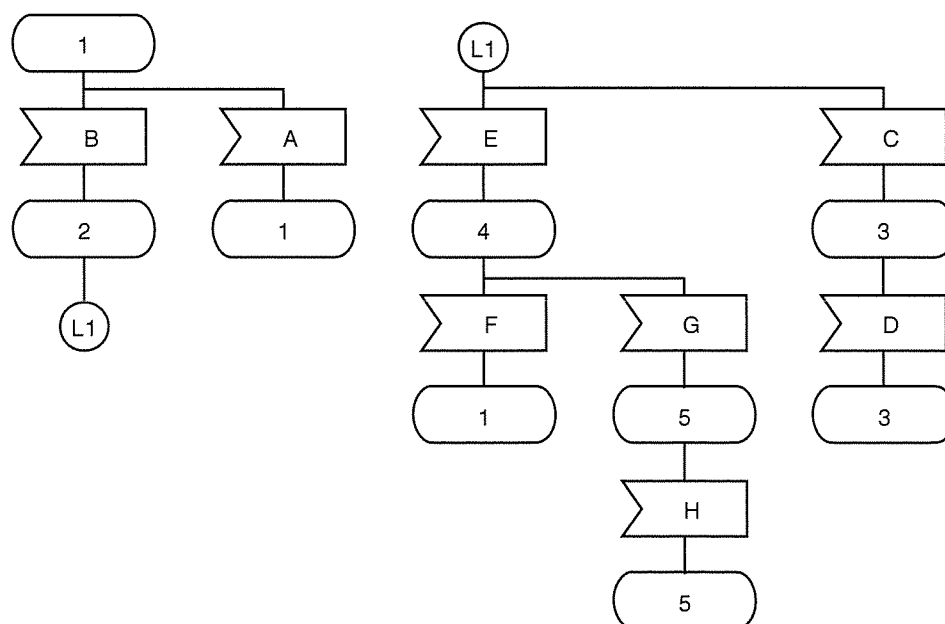


Figure 4.10 Example of a smart card state machine with the following states and transitions:

1 – initial state 2 & 4 – intermediate states 3 & 5 – final states

A – SELECT FILE

E – ASK RANDOM

B – VERIFY

F – all commands except G

C – SELECT FILE

G – EXTERNAL AUTHENTICATE

D – READ BINARY

H – SELECT FILE / UPDATE BINARY

the terminal has been successfully authenticated, the card reaches state 5. In this state, according to the diagram, files may be selected and written using SELECT FILE and UPDATE BINARY.

In this diagram, states 3 and 5 cannot be exited during a session, so they represent the two end states. A transition to state 1 is only possible via a card reset. This is not shown in the diagram, since the ‘awareness’ of every state machine is limited to its current session. No information at all is transferred from one session to the next within the state machine.

4.5 ERROR DETECTION AND CORRECTION CODES

Whenever data are transmitted or stored, it should be possible to detect any changes to the data. In particular, stored programs must be protected against corruption, since a single altered bit of program code could ruin the program or modify its execution to such an extent that it no longer provides the required functions. The EEPROM memory used in smart cards is especially sensitive to external influences, such as heat and voltage fluctuations. Consequently, the sections that perform security-related functions must be protected so that undesired changes can be detected by the operating system and their negative effects can be avoided.

Very sensitive file contents, such as program code, keys, access conditions, pointer structures and the like, must be protected against alteration. Error detection codes (EDCs) are used for

this purpose. The probability of detecting a change in a memory region protected by an EDC depends on the type of code used. Error correction codes (ECCs) are an extension of error detection. They make it possible to not only detect errors in the protected data, but to also correct errors to a limited extent

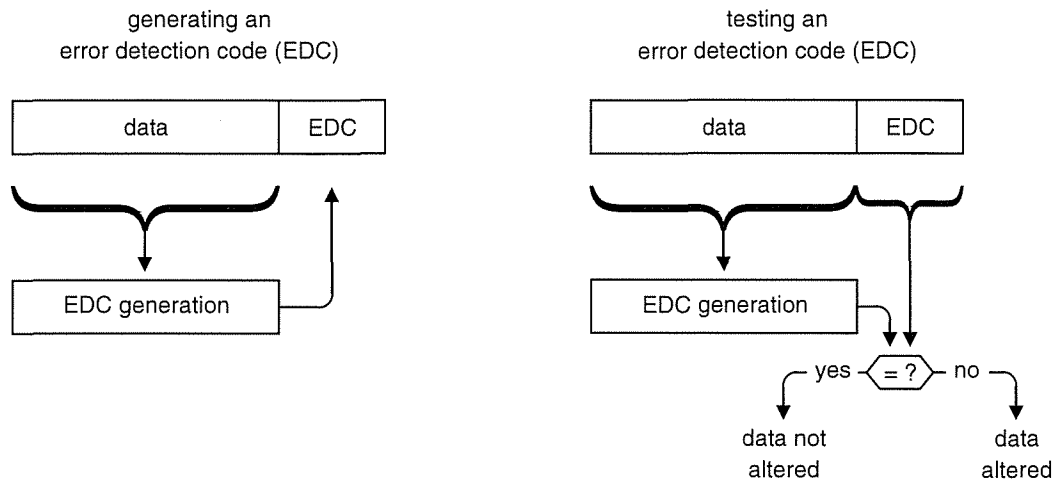


Figure 4.11 The basic processes for generating and checking an error detection code (EDC)

All of these codes work on the principle of assigning a checksum to the protected data. The checksum is usually stored along with the protected data. It is computed using a generally known algorithm, not a secret one. The data can be checked for changes as necessary by using the EDC. This is done by comparing the stored checksum with one computed anew.

A particular aspect of error detection and correction is that it utilizes a wide variety of mathematical procedures. Some of these provide a higher degree of protection for the more significant bits, in order to reduce adverse effects on numerical values as much as possible. In most cases, however, using such algorithms enormously increases the complexity and size of the program code. It is thus more common to use procedures in which error detection does not distinguish between the upper and lower parts of a byte, but instead operates on the byte as a whole.

Error detection and correction codes are very similar to message authentication codes (MACs) and cryptographic checksums (CCSs). However, there is a fundamental difference. EDC and ECC checksums can be computed and checked by anyone. In contrast, the computation of a MAC or CCS requires a secret key, since these codes are designed to protect against manipulation of the data instead of against accidental corruption.

The most widely known type of error detection code is doubtless the parity bit, which is appended to each byte in many data transmission protocols and some types of memory modules. Before computing a parity bit, you must decide whether to use even or odd parity. With even parity, the parity bit is chosen such that the total number of '1' bits in the data byte plus the parity bit is an even number. With odd parity, this total is an odd number.

If two bits in a byte are simultaneously wrong, the parity will not change and no error will be detected. Another drawback of parity-based error detection is the relatively large overhead of one parity bit for every eight data bits. This represents an additional memory load of 12.5 %. Furthermore, it is very difficult to work with supplementary parity bits when the memory is

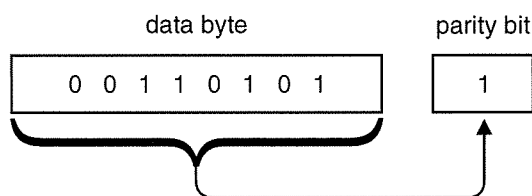


Figure 4.12 Example of error detection using a supplementary odd parity bit

organized in bytes, since this requires significant program overhead. This is why parity bits are not used for error detection in smart card memories. Other methods, such as XOR and CRC checksums, are better suited to this task.

4.5.1 XOR checksums

An XOR checksum, which is also known as a longitudinal redundancy check (LRC) due to how it is computed, can be obtained very simply and very quickly. These are both important criteria for error detection codes used in smart cards. In addition, the algorithm can be implemented extremely easily. Besides protecting data stored in memory, XOR checksums are typically used for data transmission (e.g., ATR with the T = 1 transmission protocol). An XOR checksum is computed by performing consecutive logical XOR operations on all data bytes. In other words, byte 1 is XOR-ed with byte 2, the result of this is XOR-ed with byte 3 and so on.

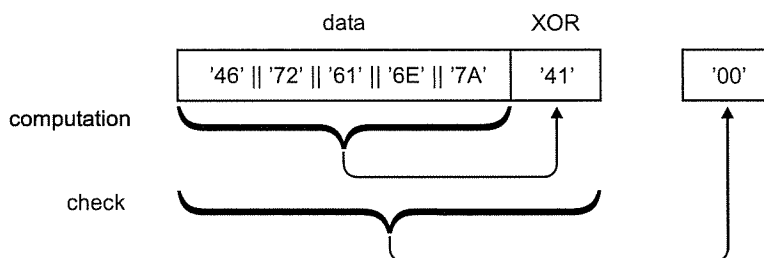


Figure 4.13 Computing and checking an XOR checksum

If the checksum is placed directly after the data and a new checksum is computed using both the data and the first checksum, the result is '00'. This is the simplest way to verify that the data and the checksum still have their original values and thus are uncorrupted.

The primary advantage of XOR checksums is that they can be computed quickly using a simple algorithm. The algorithm is so simple that its assembler code is only 10 to 20 bytes long. One reason for this is that the XOR operation is directly available in all processors as a machine instruction. In addition, an algorithm for XOR checksum computation must be implemented in almost every smart card operating system, due to the requirements of various ISO standards relating to data transmission using the T = 1 protocol. This algorithm can be used for other purposes without any additional overhead.

Unfortunately, XOR checksums also suffer from several serious drawbacks, which considerably limit their practical application. They are in principle not very secure. For example,

they do not allow the interchange of two bytes within the overall data to be detected. Also, multiple errors can occur at the same position in several bytes and cancel each other out. The consequence of all this is that XOR checksums are mainly used for data transmission; they are used only to a very limited extent to verify the consistency of memory contents. XOR checksums can be computed very quickly, since specific machine instructions for this purpose are generally available. With a typical 8-bit smart card microcontroller clocked at 3.5 MHz, a rate of 1 μ s/byte can be achieved, which corresponds to a throughput of 1 MB/s.

4.5.2 CRC checksums

The CRC method (cyclic redundancy check) also comes from the field of data communications, but it is significantly better than the XOR method. Still, a CRC checksum is also only an error detection code, so it cannot be used for error correction. The CRC method has been used for a long time in data transmission protocols, such as Xmodem, Zmodem and Kermit, and it is widely used in hard disk drive controllers in a hardware implementation. It is based on the CCITT V.41 recommendation. An additional standard for CRC checksums is ISO/IEC 13 329.

A CRC-16 checksum is generated by a 16-bit cyclic feedback shift register, while a 32-bit shift register is used to generate a CRC-32 checksum. The following description refers only to the CRC-16 checksum, since this is the most commonly used CRC checksum method for smart cards. The feedback in the shift register is determined by a generating polynomial. In mathematical terms, the data to be checked are represented as a large number, which is divided by the generating polynomial. The remainder from this division is the checksum. The CRC-16 method (that is, a 16-bit CRC) should only be used with data volumes up to 4 kB, since the error detection probability drops sharply beyond this point. However, this restriction can easily be circumvented by dividing the data into blocks that are no larger than 4 kB. Alternatively, a CRC-32 method (32-bit CRC checksum) can be used, which allows single-bit errors to be detected in up to 4 GB of data.

Table 4.6 Commonly used generator polynomials for CRC-16 computation

Designation	Generator polynomial
CRC CCITT V.41, ISO/IEC 3309	$G(x) = x^{16} + x^{12} + x^5 + 1$
CRC-16	$G(x) = x^{16} + x^{15} + x^2 + 1$
CRC-12	$G(x) = x^{12} + x^{11} + x^3 + x^2 + x + 1$

With a CRC checksum, it is always necessary to know the generating polynomial as well as the initial value for the shift register, since otherwise the computation cannot be reproduced. In the overwhelming majority of cases (e.g. ISO 3309), the initial value for the shift register is zero, but several data transmission protocols (such as CCITT Recommendation X.25) set all bits to 1.

The computation of a CRC checksum, as illustrated in Figure 4.14, proceeds as follows: (a) the 16-bit CRC register is set to its initial value; (b) the data bits are fed into the feedback shift register one after the other, starting with the least-significant bit; and (c) the feedback (which represents the polynomial division) takes place via bitwise logical XOR operations on

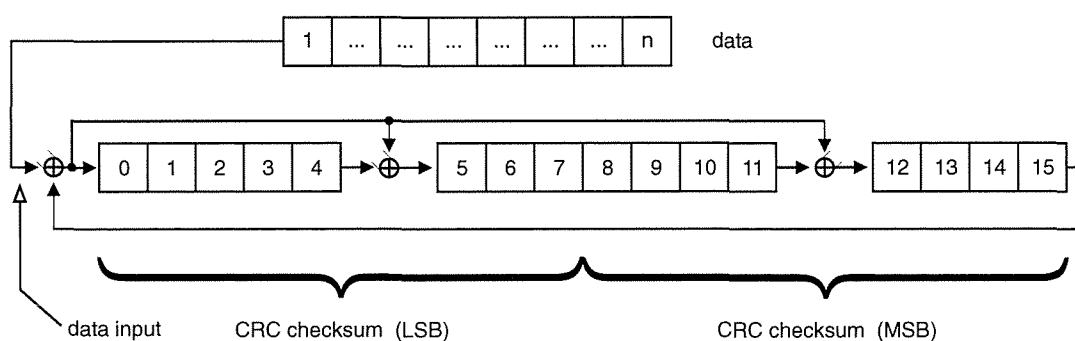


Figure 4.14 Calculating a CRC-16 checksum with a generator polynomial $G(x) = x^{16} + x^{12} + x^5 + 1$. The data and the CRC register are both shown as bits

the CRC bits. After all data bits have been fed into the register, the computation is complete and the content of the 16 bits is the desired CRC checksum.

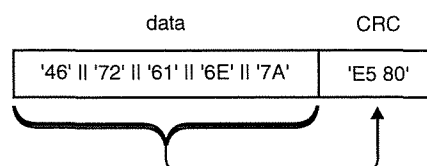


Figure 4.15 Sample computation of a CRC checksum using the generator polynomial $G(x) = x^{16} + x^{12} + x^5 + 1$ and an initial value of '0000'

A CRC checksum can be verified by again calculating the CRC checksum of the data and comparing the result with the checksum provided with the data. If they are the same, it follows that the data and the checksum have not been altered.

The major advantage of CRC checksums is that they provide reliable error detection, even with multiple errors. Only very few methods can achieve this. In addition, in contrast to the XOR method, CRC allows interchanged data bytes to be detected, since byte order definitely plays a role in checksum generation via the feedback shift register. However, it is very difficult to specify exact detection probabilities for such errors, since they are very dependent on the locations of the errors within the bytes in question.

The CRC algorithm is relatively simple, and the amount of code needed to implement it thus matches the needs of small smart card memories. Its greatest drawback is the slowness of the computation, since the algorithm requires the data to be shifted bit by bit. The CRC checksum algorithm was originally designed for hardware implementation, and this has a strong detrimental effect when it is implemented in software. The throughput of a CRC-16 routine is lower than that of an XOR checksum routine by a factor of around 200. A typical figure is 0.2 ms/byte at a 3.5-MHz clock frequency, which corresponds to a throughput of 5000 byte/s. Computing a CRC checksum for a 10-kB smart card ROM would thus require around 2 seconds.

Many types of microcontrollers have a special component for hardware-assisted generation of CRC checksums for definable memory regions. The rates that can thereby be achieved can be as high as one byte per clock cycle. For a microcontroller clocked at 5 MHz without an internal clock divider, this would yield a throughput of 5 MB/s.

4.5.3 Reed–Solomon codes

In 1960, the mathematicians Irving S. Reed and Gustave Solomon published a paper with the title ‘Polynomial Codes over Certain Finite Fields’, which forms the basis for what has become one of the most widely used methods for error detection and correction. Reed–Solomon codes (RS codes), which are named after their inventors, are used for error detection and correction for data storage (e.g. bar codes, DAT, CD and DVD) and data transmission (e.g. DSL, satellite communications and space probes).

Reed–Solomon codes are block-oriented error correction codes (ECCs) that can also correct burst errors. They are computed using the arithmetic of finite bodies (Galois fields, or GF). The characteristics of a Reed–Solomon code, in terms of the number of detectable and correctable errors for a specific data length, can be adapted to a particular application via the choice of generator polynomial.

For several years now, Reed–Solomon codes have been used by various smart card operating systems for error detection, and quite rarely for error correction, in order to secure data stored in EEPROM. The generator polynomials used are matched to the properties of EEPROM storage with regard to the occurrence of burst errors, thus yielding significantly more secure error detection than what is possible using the CRC method.

For example, with an RS code using a 2^8 GF and two supplementary bytes in addition to the data to be protected, it is possible to detect two incorrect bytes or correct one incorrect byte. With three supplementary bytes, three incorrect bytes can be detected or one incorrect byte can be corrected, and with four supplementary bytes, four incorrect bytes can be detected or two incorrect bytes can be corrected. The size of the executable code in 8051 assembler is approximately 100 bytes, and the computation rate is approximately 10 ms/byte at 3.57 MHz. RS codes are also quite suitable for implementation in hardware.

4.5.4 Error correction

If it is necessary to not only detect changes in memory regions, but also to correct them if possible if they result from errors, error correction codes must be used. Since computing such codes is costly in terms of program code, using them to protect smart card memory is problematic. Furthermore, the algorithms in question are usually designed to correct only low error rates. Since EEPROM memory in smart cards is page-oriented, a whole page usually fails in the event of an error, so only methods that are capable of correcting burst errors are worth considering. Consequently, a different approach is taken for error correction.

The technically simplest solution is to store the data in multiple, physically separate memory pages and use a majority-vote procedure when reading the data. Triple storage is commonly used, together with a 2-of-3 vote. A less memory-intensive variation of this method is to store the data in two locations with EDC checksum protection for each location. The occurrence of a memory error can be detected by checking the two EDC values. This also allows the memory region where the error occurred to be identified. The region with no detected error must then contain the valid data, which can be restored to the faulty region.

Of course, a significant amount of extra memory is needed for these error correction methods, but for small quantities of data it is still well within acceptable limits. The main advantage is that no complicated, code-intensive algorithm is needed to evaluate the data.

As an alternative to protection by multiple data storage, it is possible to use an error correction algorithm, such as the Reed–Solomon algorithm. This is particularly suitable for use with clustered errors, such as may occur in smart cards due to page failures. The algorithm occupies a few hundred bytes of memory when programmed in assembly language, and the size of the ECC data depends primarily on probability with which errors must be detected and/or properly corrected.

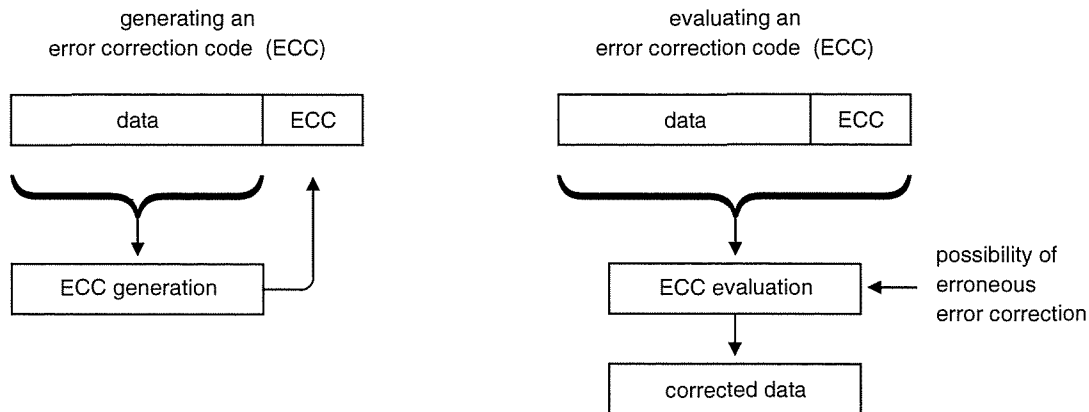


Figure 4.16 The basic principle of using an error correction code (ECC)

Several basic remarks are in order here with regard to using error correction methods in smart cards. At first glance, it may be tempting to use these methods to correct errors that occur in the EEPROM. However, the presumed data security is bought at the price of several serious drawbacks. The required amount of memory space is enormous, and the time required to write data to memory also increases considerably, since data must be stored in multiple locations. Algorithms that can correct clustered errors on the scale at which they typically occur with page-based EEPROMs are complicated and require a large amount of memory space for the EDCs. However, there is an even more serious fundamental drawback. Even when an error correction algorithm is used, errors can in principle still be present in the corrected data, since the algorithm works properly only up to a certain number of errors. If an operating system corrects memory errors automatically, in principle it is never possible to be certain that the correction was made properly.

For example, suppose that automatic error correction is applied to the balance in an electronic purse. The system operator can never be sure of what will happen to the credited amounts in the event of an error. The balance may be corrected properly, but there is a certain probability that it will be too high or too low after the correction. In this regard, it must be remembered that smart cards are inexpensive mass-produced articles, which can simply be replaced if they are faulty.

As a rule, when problems occur with data contents, a higher level system that allows human intervention must decide what to do. For example, on the first instance of an error occurring in a smart card purse, the cardholder's balance will most likely be manually restored. However, if the error recurs repeatedly, the system operator will be much less forthcoming with regard to the cardholder, since there is a possibility that the EEPROM has been fraudulently manipulated. This cannot be handled by an error correction code in the card; instead, the system administrator must intervene.

4.6 DATA COMPRESSION

As is well known, the amount of memory available in a smart card is severely limited. Consequently, the desire to improve this situation by using data compression repeatedly arises among application providers.

There are certain hurdles that must be overcome before data compression can be used. The algorithm must not take up too much memory space, and in particular it must require very little RAM. In addition, an acceptable compression speed should be achieved. The compression factor is not all that important, since the data volume is always only a few hundred bytes at most. The methods that are frequently used for smart cards are run-length encoding and variable-length encoding.

With run-length encoding, a contiguous string of identical data objects is replaced by the combination of a repetition count and the object (such as a character) to be repeated. With variable-length encoding, the frequency of occurrence of characters having a fixed length (e.g., one byte) is analyzed, and the most frequently occurring characters are replaced by characters with shorter lengths (Huffman algorithm). Less frequently occurring characters are encoded using longer codes.

With static variable-length encoding, replacements are made using a previously defined table. The dynamic version of variable-length encoding first analyzes the frequency distribution of the characters in the original data and then constructs a replacement table based on the results of this analysis. A third variation is adaptive variable-length encoding, in which the replacement table is continuously updated during the compression process to achieve optimum compression.

Both dynamic and adaptive variable-length encoding are out of the question for smart cards, due to the complexity of their algorithms and their large memory requirements. Run-length encoding and static variable-length encoding are thus the only real alternatives for use in smart cards. The algorithm for run-length encoding does not need much program code, but it has the drawback that it can only be used with repetitive data. Image data, for example, are particularly suitable, since images often contain large areas with the same value. Keys for symmetric cryptographic algorithms would be completely unsuitable for compression with this algorithm, since they have the characteristics of random numbers.

Static variable-length encoding is the second compression method used with smart cards. It is quite suitable for files containing telephone directory information, for instance, since the structure of the stored data is known and the replacement table can be permanently built into the algorithm. Telephone numbers consist of only the numerals 0 through 9 and a few special characters, such as '*' and '#'. If only capital letters are allowed for names, the replacement table only has to accommodate the 26 characters of the alphabet. Furthermore, certain letters occur significantly less often in names than do others, which also affects the encoding. With telephone directories, a memory space reduction of 30 % (compared with the uncompressed data) can certainly be achieved, although this does not take into account the memory occupied by the compression algorithm.

However, certain things must be considered with regard to data compression for smart cards. Ideally, data compression should be performed in the operating system in a manner that is fully transparent to the outside world, such that uncompressed data can be read and written in the usual way using standard commands. Compression can also only be applied to certain types of data. The results of attempting to compress program code and keys are usually unsatisfactory. This must be taken into account in the design of the application, since otherwise the anticipated

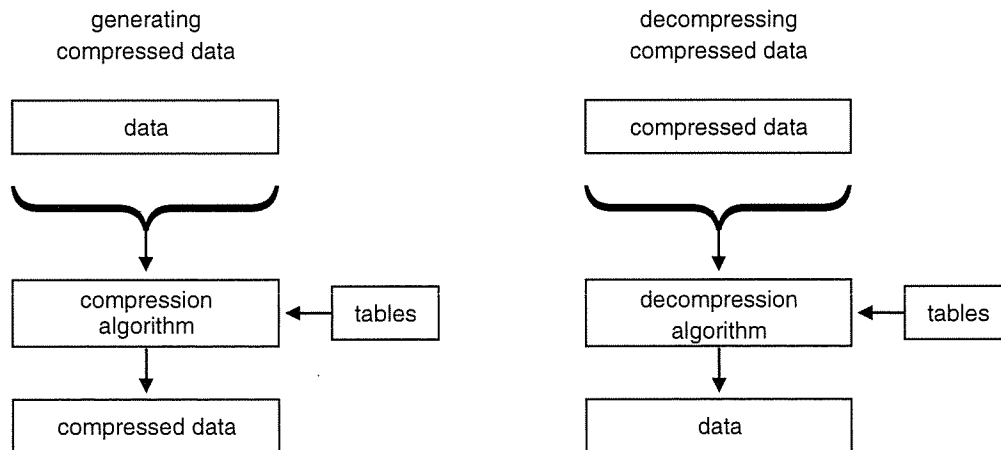


Figure 4.17 The basic principle of data compression for stored data

reduction in memory space can (in the worst case) turn into a need for even more memory as the result of ‘compression’.

For all of these reasons, data compression has been used only sparingly in smart cards up to now. In special applications, such as telephone directories in cards used in the telecommunications sector, compression algorithms are sometimes used. With general-purpose operating systems and applications in which the structure of the data is not known in advance, using data compression does not produce satisfactory results. It should thus be avoided, due to the additional memory space required by the compression algorithm.

4.7 CRYPTOLOGY

In addition to their function as data storage media, smart cards are also used as authorization media and encryption modules. As a result, cryptography achieved a central significance in the early days of smart cards. Nowadays, the procedures and methods of this discipline are firmly established components of smart card technology.

Cryptology can be split into two areas of activity, namely cryptography and cryptanalysis. Cryptography is the study of the methods used for encrypting and decrypting data, while cryptanalysis is concerned with attempting to break existing cryptographic systems.

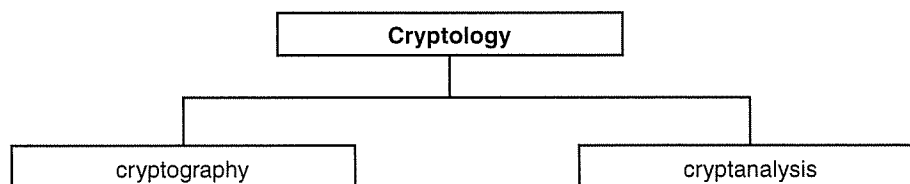


Figure 4.18 The two subdivisions of cryptology are cryptography and cryptanalysis

In the smart card realm, the practical use of existing cryptographic procedures and methods represents the principal task and primary application area with regard to cryptography. Consequently, here we concentrate more on the practical aspects of cryptography than on the

theoretical aspects. However, we do not entirely neglect the application of the procedures and the basic features of their theoretical foundations.

The four objectives of cryptography are maintaining the secrecy of messages (*confidentiality*), ensuring the *integrity* and the *authenticity* of messages and ensuring the binding force (*non-repudiation*) of messages. These objectives are mutually independent, and they place different demands on the system in question. Confidentiality means that only the intended recipient of a message can decrypt its contents. Authenticity means that the recipient can verify that the received message has not been altered in the course of being transmitted. Non-repudiation means that the sender can verify that a certain recipient has received a particular message, which means that the message has binding force.

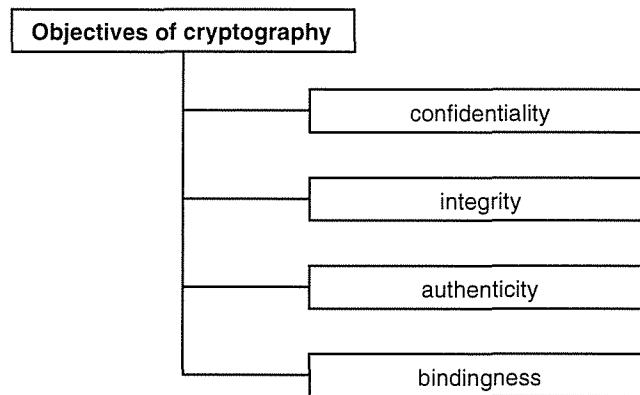


Figure 4.19 Classification of the four independent objectives of cryptography

The notation used in this book for cryptographic procedures is illustrated in Figures 4.20 and 4.21. The terms and principles described below form the basis for cryptology and are a prerequisite for understanding the procedures described in the rest of this section.

In simplified terms, there are three types of data in encryption technology. The first is *plaintext*, which is unencrypted data. Encrypted data is referred to as *ciphertext*. Finally there is a *key*, one or more of which is required for encryption and decryption. These three types of data are processed by an encryption algorithm. The algorithms that are currently used in smart cards are generally block-oriented, which means that the plaintext and ciphertext can only be processed in packets with fixed lengths (such as 8 bytes with DES).

Modern cryptographic algorithms are generally based on Kerckhoff's principle. This principle, which is named after Auguste Kerckhoff (1835–1903), says that the entire security of an algorithm should be based only on the secrecy of the key, and not on the secrecy of the cryptographic algorithm. The consequence of this generally known but often-disregarded principle is that many algorithms used in the civil sector have been published, and in part also standardized.

The opposite of Kerckhoff's principle is the principle of security by concealment. With this principle, the security of a system is based on the idea that a would-be attacker does not know how the system works. This principle is very old, and it is still frequently used even today. However, you should take care not to develop a cryptographic system (or any other system) based on this principle alone. Up to now, every system based on this principle alone has been broken, usually in a very short time. In our information society, it is generally not possible to

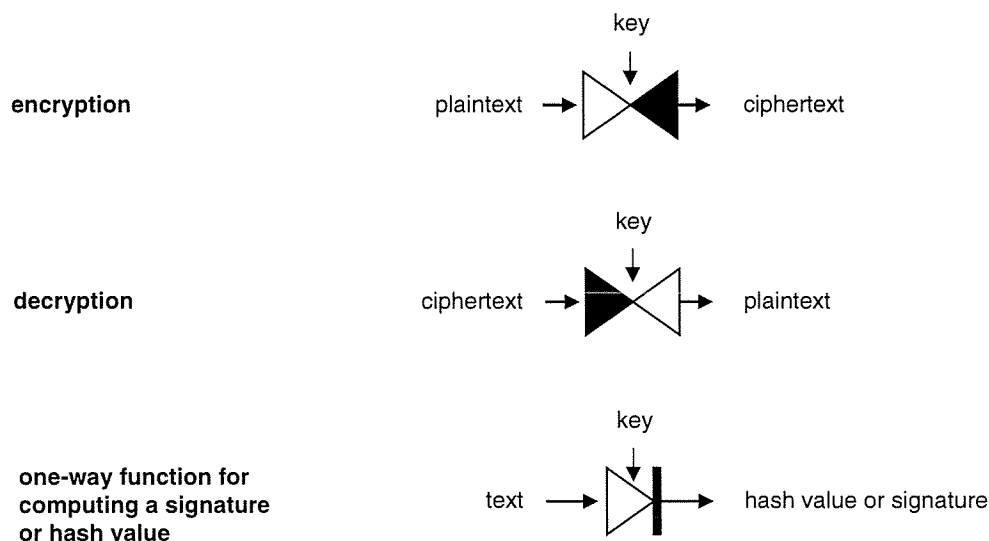


Figure 4.20 The symbols used in this book for cryptographic algorithms

keep the technical details of a system secret for a long time, and that is precisely what this principle requires.

Of course, the consequences of the unintentional interception of messages can most certainly be limited by using concealment. This principle is thus repeatedly used in combination with Kerckhoff's principle. In many large systems, it is also incorporated as a supplementary security level. Since the security of modern, published cryptographic algorithms is primarily based only on the limiting processing capacity of current computers, concealing the procedure that is used increases the level of protection against attacks.

If you rely only on the protection provided by the assumption that a potential attacker does not have access to sufficient processing power, you may be quickly overtaken by the rapid pace of technical progress. Statements such as 'it would take a thousand years to break this cryptographic system' are unreliable, since they are based on currently available processing capacities and algorithms. They cannot take future developments into account, since such developments are generally unknown. The arithmetic processing capacities of processors double around every 18 months, which means that the capacity per processor has increased by a factor of approximately 25,000 over the last 25 years.

Recently, the increased degree of networking of computers has created another option for mounting serious attacks on keys or cryptographic systems. For instance, a request to help break a DSS key, if posted on the Internet, would be forwarded to millions of users by the snowball effect. If only 1 % of all current users² participated in such an action, the potential attacker would have access to a parallel computer composed of 300,000 individual computers.

Cryptographic algorithms are divided into two types: *symmetric* and *asymmetric*. This division is based on the key that is used. Here 'symmetric' means that the algorithm uses the same key for encryption and decryption. By contrast, asymmetric algorithms (which were

² In the summer of 2001, it was assumed that there were around 500 million users of the Internet

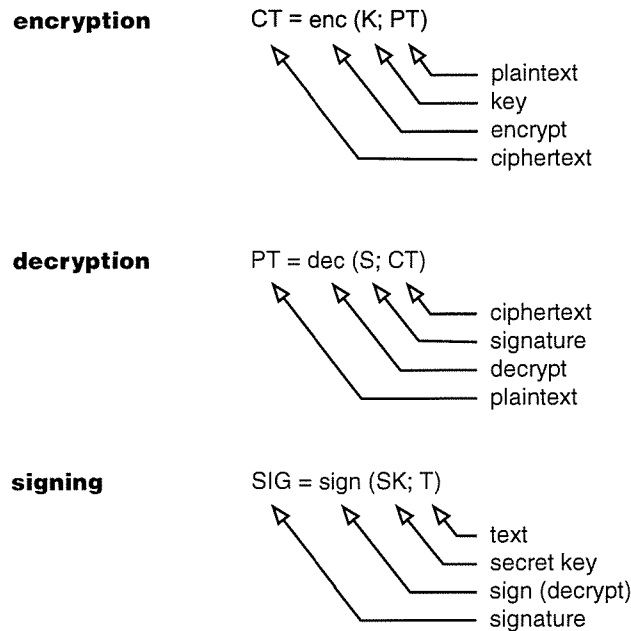


Figure 4.21 The notation used in this book for cryptographic procedures

postulated in 1976 by Whitfield Diffie and Martin E. Hellman) use different keys for encryption and decryption.

A term that often comes up in connection with cryptographic algorithms is the magnitude of the key space. This refers to the number of possible keys that can be used with a particular cryptographic algorithm. A large key space is one of several criteria for a secure cryptographic algorithm.

A requirement that has only recently become prominent with regard to the technical implementation of cryptographic algorithms in smart cards is freedom from noise. In this context, this means that the execution time of the algorithm must not depend on the key or the plaintext and ciphertext. If this requirement is not met, it could be possible to discover the key in a relatively short time, which would mean that the entire cryptographic system was broken.

In cryptology, there is a strong distinction between the theoretical and practical security of a system or an algorithm. A system is *theoretically* secure if an attacker, given unlimited time and technical resources, cannot break the system. This means that even if an attacker would need 100 years and the aid of several supercomputers to break a system, it could not be considered to be theoretically secure. If a system cannot be broken when the attacker has only a limited amount of time and technical resources, it is considered to be *practically* secure.

A cryptographic system can assure the confidentiality and/or authenticity of a message. If the system has been broken, this means that confidentiality and/or authenticity are no longer guaranteed. If an attacker can discover the secret key of an encryption algorithm, for example, he can then decrypt data that have been protected by being encrypted, in order to learn their content and modify them as desired.

Several different methods of attack can be used to break the key of a cryptographic algorithm. In a 'ciphertext-only' attack, the attacker knows only the ciphertext and attempts to determine

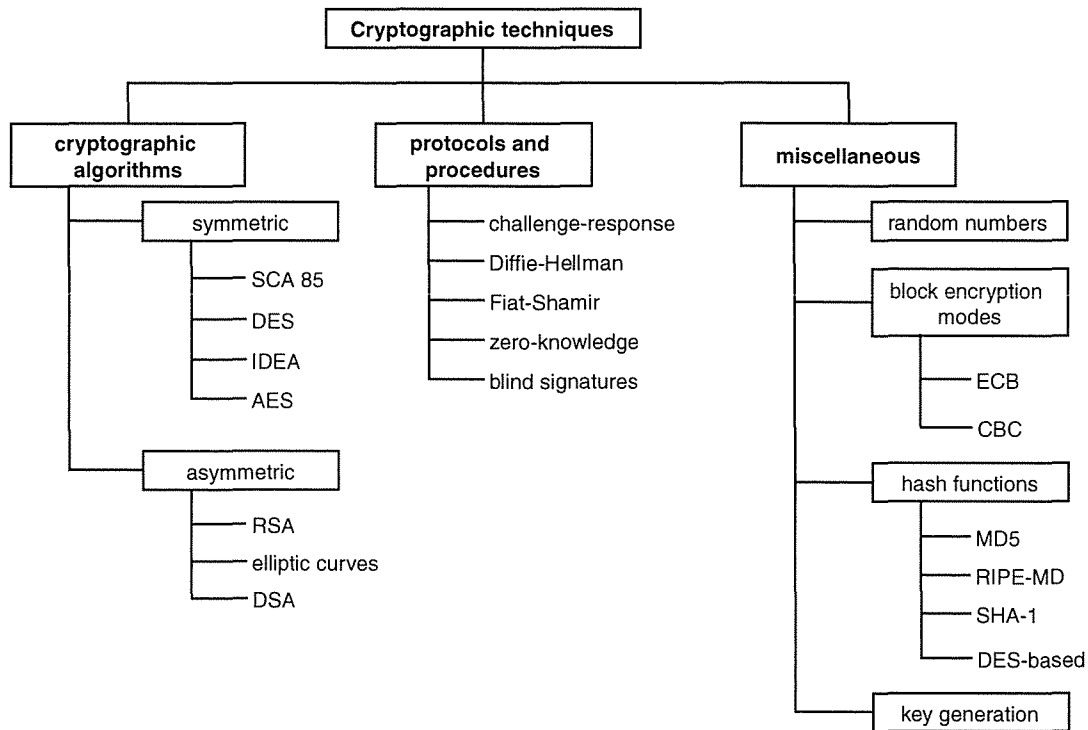


Figure 4.22 Classification chart for cryptographic techniques used in the smart card area

the key or plaintext from the ciphertext. A more promising method of attack is the 'known-plaintext' attack, which involves the attacker knowing several plaintext-ciphertext pairs for a secret key. The 'chosen-plaintext' and 'chosen-ciphertext' attacks require the attacker to be able to generate his own plaintext-ciphertext pairs. If this is possible, the likelihood of success is improved, since the secret key can be discovered experimentally.

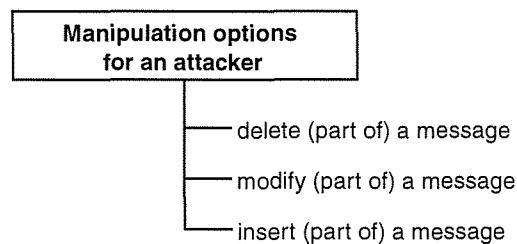


Figure 4.23 Classification of the various manipulation options available to an attacker

Discovering a key by trial and error (a 'brute-force' attack) is naturally the least sophisticated method of attack. With this method, an attempt is made to find out the correct key by employing a large amount of processing capacity to test all possible keys with a known plaintext-ciphertext pair. Obviously, a processing capacity in the supercomputer range is normally a prerequisite for this method. Statistically seen, on average half of the possible keys must be tested before the right one is found. Naturally, a large key space considerably increases the difficulty of such an attack.

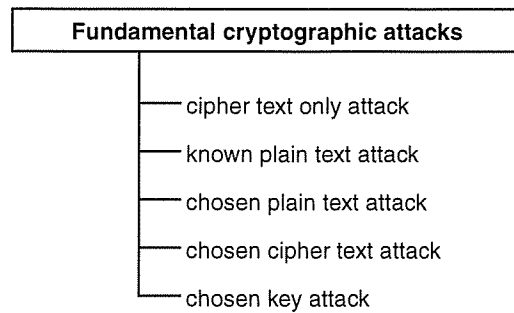


Figure 4.24 Classification of the basic methods of cryptographic attack

4.7.1 Symmetric cryptographic algorithms

Symmetric cryptographic algorithms are based on the principle of performing encryption and decryption using the same secret key – hence the designation ‘symmetric’.

The DES algorithm

The best-known and most widely used type of symmetric cryptographic algorithm is the Data Encryption Algorithm (DEA). It was developed by IBM in collaboration with the NBS (US National Bureau of Standards) and published in 1977 as the US FIPS 46 standard. The standard that describes the DEA is generally referred to as the DES (Data Encryption Standard). Consequently, the Data Encryption Algorithm is often (but not entirely correctly) called the DES algorithm.

Since this algorithm is of course fashioned according to Kerckhoff’s principle, it could be published without losing any of its security. However, even today not all of the development criteria have been made known, which repeatedly leads to assumptions regarding possible methods of attack and the possible presence of ‘trap doors’. However, up to now all attempts to break the algorithm on this basis have failed.

Two important principles for a good encryption algorithm were incorporated into the design of the DES algorithm. These are the principles of *confusion* and *diffusion*, as first proposed by C. Shannon. The confusion principle states that the statistics of the ciphertext should affect the statistics of the plaintext in a manner that is so complex that an attacker can derive no advantage from them. The diffusion principle states that each bit of the plaintext and each bit of the key should affect as many bits of the ciphertext as possible.

DES, which is a symmetric block encryption algorithm, does not expand the ciphertext. This means that the plaintext and ciphertext blocks have the same size. The block size is 64 bits (8 bytes), which is also the length of the key, although only 56 of these bits are used as the actual key. The key contains eight parity bits, which reduces the available key space. The 64 bits of the key are numbered consecutively from left (msb) to right (lsb). Bits 8, 16, 24, ..., 64 are the parity bits. The parity is always odd. Due to the parity bits, the key space is 2^{56} , which means that there are approximately 7.2×10^{16} possible keys. At first glance, a key space with 72,057,594,037,927,936 possible keys may appear very large, but the limited size of its

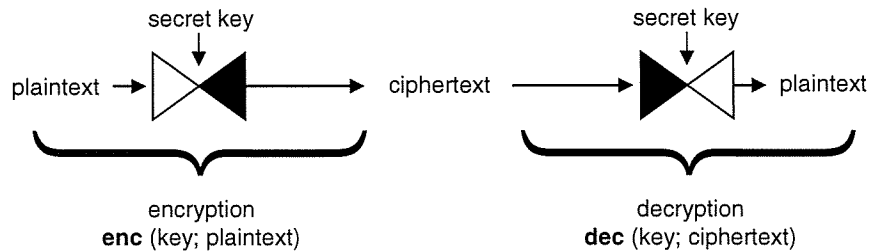


Figure 4.25 Operating principle of a symmetric cryptographic algorithm for encrypting and decrypting data. The DES is a typical example of this type of cryptographic algorithm

key space is actually the main weakness of the DES algorithm.³ Given the steadily increasing processing capacity of modern computers, a key space of this size is already considered to be too small for a secure cryptographic algorithm. If a plaintext–ciphertext pair is available, it is easy to test all possible keys if the key space is too small.

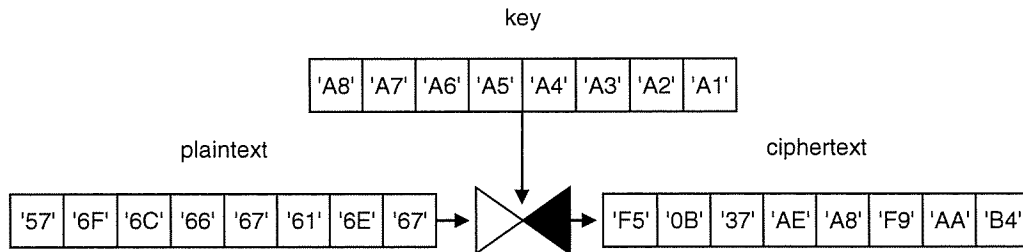


Figure 4.26 The operation of the DES algorithm for encrypting data

If a plaintext–ciphertext pair is obtained by tapping the communications between a terminal and a smart card, a brute-force attack can be mounted by encrypting the plaintext using all possible keys. The correct key can be determined by comparing all of the resulting ciphertexts with the previously obtained ciphertext. This procedure can very easily be executed using several parallel processors. Each of the processors tests only the relatively small portion of the key space assigned to it. A sample calculation can illustrate the amount of time needed for such a brute-force attack. The fastest currently available DES components require 64 ns for a complete block encryption.⁴ If 10,000 such computation modules are assembled in parallel, each one can independently test a small part of the key space. Assuming that on average only half of the key space must be searched to find the correct key, the processing time can be calculated as follows:

$$((2^{56} \times 64 \text{ ns}) \div 10,000) \times 0.5 \approx 64 \text{ h}$$

In 1993, Michael Wiener published plans for a million-dollar computer that could test all the DES keys of a given plaintext–ciphertext pair within seven hours [Wiener 93]. In 1998, a

³ To get a feeling for the size of this large number, you might enjoy considering the following comparison. The mass of the earth is roughly 5.974×10^{27} grams. Based on this value, the number of electrons, protons and neutrons that make up the earth can be taken to be around 10^{52} . If only a single elementary particle were needed to store a bit, then at most 10^{52} bits could be stored in a memory with the mass of the earth

⁴ Using a DEC gate array built with gallium-arsenide technology that operates in the ECB and CBC modes

similar concept, called ‘DES Cracker’, was implemented by the Electronic Frontier Foundation (EFF) at a cost of approximately US \$ 250,000. The DES Cracker can test approximately 88.8 billion keys per second, and in a public test it took 56 hours to determine an unknown key [EFF 98]. The biggest problem with the use of DES is that its key space has become too small over time. Consequently, the triple-DES algorithm is used almost exclusively for new applications.

A full description of the exact implementation of the DES algorithm is far beyond the scope of this book, and it is not necessary for understanding the subject. If you are interested in having more detailed information, consult FIPS Publication 46, Carl Meyer [Meyer 82] or Bruce Schneier [Schneier 96]. However, one significant detail must be mentioned. The DES was designed as an encryption algorithm that can easily be implemented in hardware. There are presently many smart card microcontrollers with a DES hardware module. However, if it is necessary to implement DES in a smart card in software, it will occupy around 1 kB of assembler code, even in a highly optimized version. The size of a DPA-resistant version is approximately 2 kB. Its computational speed is consequently rather low.

Typical computation times for encryption and decryption using a smart card, with comparison values for using a PC and a hardware integrated circuit, are shown in Table 4.7. These numbers can vary depending on the actual implementation, and they take into account only the pure processing time for DES encryption or decryption of an 8-byte block, assuming that all registers are already loaded. As a general rule, it can be assumed that DES implemented in hardware is approximately 150 times faster than DES in software.

Keys for the DES algorithm can be generated using a random number generator that produces an 8-byte random number, which is then checked against the four weak and 12 semi-weak keys. If the computed value does not match any of these easily broken keys, the parity bits are computed and the result is a DES key.

Table 4.7 Typical DES computation times (8-byte block)

Implementation	Computation time / throughput
Smart card, 3.5-MHz clock, software implementation	17.0 ms / 3.8 kbit/s
Smart card, 3.5-MHz clock and DES processing unit	112 μ s / 571 kbit/s
Smart card, 3.5-MHz clock and triple-DES processing unit	130 μ s / 492 kbit/s
Smart card, 4.9-MHz clock, software implementation	12.0 ms / 5.3 kbit/s
Smart card, 4.9-MHz clock and DES processing unit	80 μ s / 800 kbit/s
Smart card, 4.9-MHz clock and triple-DES processing unit	93 μ s / 688 kbit/s
PC (80486, 33 MHz)	30 μ s / 2.1 MB/s
PC (Pentium, 90 MHz)	16 μ s / 4 MB/s
PC (Pentium, 200 MHz)	4 μ s / 16 MB/s
DES integrated circuit	64 ns / 100 MB/s

IDEA algorithm

There are many other symmetric cryptographic algorithms besides DES. Here we consider only the International Data Encryption Algorithm (IDEA) as a representative example. It was developed by Xuejia Lai and James L. Massey, and it was published in 1990 as the ‘proposed

encryption standard' (PES). It was improved in 1991, and for a short while the improved version was known as the 'improved proposed encryption standard' (IPES), but nowadays it is commonly known as IDEA. The development criteria and internal construction of this algorithm have been fully published, so Kerckhoff's principle is satisfied. However, IDEA is subject to patent restrictions, which was formerly also the case with the RSA algorithm.

IDEA, like DES, is a block-oriented cryptographic algorithm, and it also uses 8-byte plaintext and ciphertext blocks. In contrast to the DES algorithm, the key length is 16 bytes (2×8 bytes). This provides a significantly larger key space, with a size of $2^{128} \approx 3.4 \times 10^{38}$. In regular decimal notation, the number of possible keys for the IDEA is exactly 340,282,366,920,938,463,463,374,607,431,768,211,456.

Due to its structure, IDEA is compatible with DES except for its extended key length. It is also compatible with triple-DES systems, which use keys that are 2×56 bits long, which means that changing the algorithm used does not affect the lengths of the keys or the input and output data blocks. Of course, compatibility in this regard does not mean that DES-encrypted data can be decrypted using the IDEA. In general, IDEA is considered to be a very good cryptographic algorithm. It has also been widely distributed in the form of the public-domain program PGP (Pretty Good Privacy) from Philip Zimmermann, which is used for secure data transmission.

There are only a few smart card implementations of IDEA. The amount of memory space needed for the program is around 1000 bytes. Typical computation times for encryption and decryption are somewhat less than for DES. However, in the development of IDEA, it was assumed that the computations would be executed by a 16-bit processor. Since most smart cards still have 8-bit processors, the speed advantage in comparison with DES is not as great as might be expected. Table 4.8 lists typical values for IDEA operations on an 8-byte block, assuming that previously computed keys are available.

Table 4.8 Typical IDEA computation times (8-byte block)

Implementation	Computation time
Smart card with 3.5-MHz clock, software implementation	12.3 ms
Smart card with 4.9-MHz clock, software implementation	8.8 ms
PC (80386, 33 MHz)	70 μ s
PC (Pentium Pro, 180 MHz)	4 μ s
IDEA integrated circuit	370 ns

AES algorithm

The amount of processing capacity available at the end of the 1990s, as the result of technical progress and the international networking of computers, was sufficient to allow even ambitious private individuals with organizational abilities to mount successful brute-force attacks on the DES algorithm. As a result, the future viability of DES became so questionable that the relevant national authorities increasingly devoted their attention to specifying a new symmetrical cryptographic algorithm as the official successor to DES. In light of the never fully extinguished doubts about the integrity of DES arising from the fact that not all of the design criteria were made available for public inspection, in 1997 the US National Institute

of Standards and Technology (NIST) requested the international cryptographic community to submit possible successors to DES, accompanied by all associated documentation, to the NIST for a sort of competitive evaluation. In 1998, the NIST announced that there were 15 candidates for the Advanced Encryption Standard (AES) and published all of the documentation so that it could be studied with regard to informatics and cryptanalytical aspects by the US National Security Agency (NSA), research institutes, experts and interested parties. Following detailed investigations of these methods and broad public discussion of the results, a candidate for the successor to the DES algorithm was announced by the NIST in 2000.⁵ This algorithm was developed by the Belgian cryptologists Joan Daemen and Vincent Rijmen and was already known as the Rijndael algorithm prior to the announcement of the competition.

AES is a symmetric block encryption algorithm with a block length of 128 bits (16 bytes) that can be used with three different key lengths: 128 bits (16 bytes), 192 bits (24 bytes) and 256 bits (32 bytes). It is thus referred to as AES-128, AES-192 or AES-256, depending on the key length. AES can be implemented in hardware logic, and good software implementations are also possible using relatively low-performance 8-bit processors as well as high-performance 16- and 32-bit processors. Furthermore, it can be used throughout the world free of licensing fees, and according to the official pronouncement its anticipated useful life is more than 20 years. AES is standardized by FIPS 197, which is available free of charge via the Internet [NIST].

The size of the key space of the AES algorithm with a 128-bit key is approximately 3.4×10^{38} (2^{128}), which makes it a factor of 4.7×10^{21} greater than the key space of DES with a 56-bit key. The large key space obtained with a key length of only 128 bits enormously increases the difficulty of attacks involving successively testing all possible keys. The software implementation of AES in a DPA-resistant form in a smart card occupies approximately 4 kB of ROM.

Table 4.9 Typical computation times for AES using a 128-bit key (16-byte block)

Implementation	Computation time
Smart card, 16-bit CPU, 4.9-MHz clock, software implementation; encryption	20 ms
Smart card, 16-bit CPU, 4.9-MHz clock, software implementation; decryption	25 ms

Operating modes for block-oriented encryption algorithms

The DES algorithm, like every block-oriented encryption algorithm, can be used in four different operating modes that are standardized in ISO 8372. Two of these operating modes, the CFB and OFB modes, are especially suitable for sequential text with no block structure. The other two, the ECB and CBC modes, are based on a block size of 8 bytes. These two block-oriented modes are most commonly used in smart card applications.

The basic operating mode of the DES algorithm is designated as the ECB (electronic code book) mode. In this mode, 8-byte plaintext blocks are independently encrypted using a single key. This is the DES algorithm in its pure form, without extensions.

⁵ A good summary of the selection process and selection criteria can be found in [Nechvatal 00]

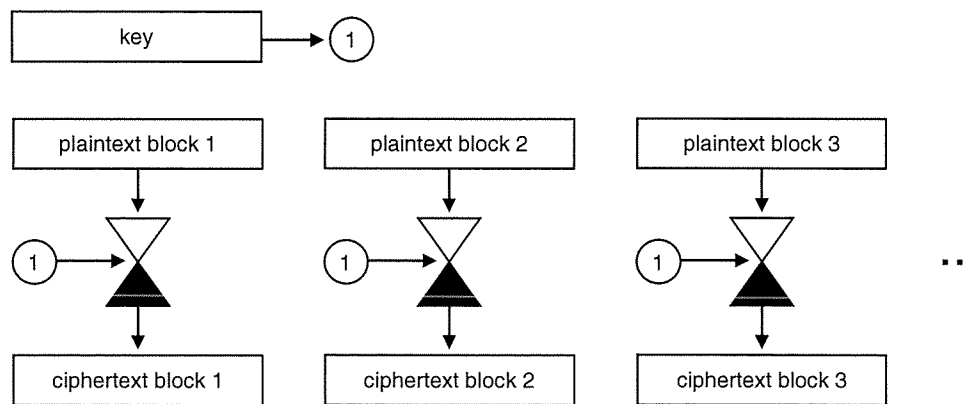


Figure 4.27 Data encryption with a block-oriented encryption algorithm operating in the ECB mode. Decryption is performed in a similar manner

The second block-oriented mode is known as the CBC (cipher block chaining) mode. In this mode, a data string consisting of several blocks is chained using XOR operations during the encryption such that each block becomes dependent on the block preceding it. This makes it possible to reliably detect interchanging, adding or deleting encrypted blocks. This is not possible with the ECB mode. If the plaintext blocks are suitably structured (with sequence counters in their headers or initialization vectors), a consequence of CBC chaining is that even identical plaintext blocks are converted into non-identical ciphertext blocks. This makes the cryptanalysis of intercepted data much more difficult, since codebook analysis (for example) is not possible.

The first plaintext block is XOR-ed with an initialization vector (often called an IV), and then encrypted with the DES algorithm. The result is the ciphertext, which is in turn XOR-ed with following plaintext block. The process continues in this manner with the following blocks. As a rule, the initialization vector is preset to null. However, in some systems, a session-specific random number is written to the initialization vector as a substitute for a temporary key. This number must naturally be known when the data are subsequently deciphered.

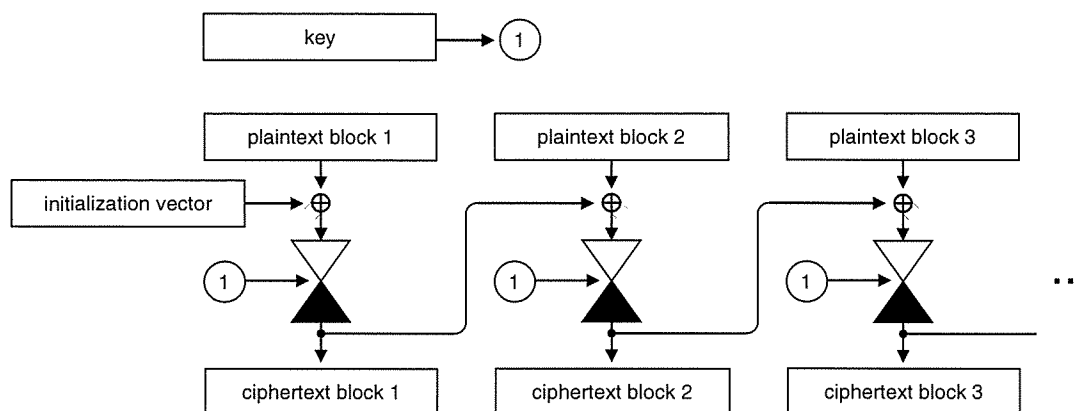


Figure 4.28 Data encryption with a block-oriented encryption algorithm operating in the CBC mode. Decryption is performed in a similar manner

Multiple encryption

In addition to the four operating modes of a block-oriented encryption algorithm, another variation is also used to improve security. However, it is actually only used with the DES algorithm, due to the small key space of this algorithm. In principle, it can be used with any block-oriented encryption algorithm that is not a group. If an encryption method has the group property, double encryption using two different keys will not increase the level of security, since the same result could be obtained by single encryption using a third key. The size of the key space is not increased by double encryption using an algorithm that has the group property, since an attacker would only have to discover the third key in order to obtain the result previously obtained by double encryption using two different keys:

$$\text{ciphertext} = \text{enc}(\text{key } 2; (\text{enc}(\text{key } 1; \text{plaintext})))$$

Since DES is not a group, the result of double DES encryption using two different keys fundamentally cannot be duplicated by single encryption using a third key.

However, in 1981 Ralph C. Merkle and Martin E. Hellman published a method of attack called the ‘meet-in-the-middle’ attack [Merkle 81], which can be used very successfully against every type of double encryption using a block-oriented encryption algorithm. It presupposes that the attacker knows several plaintext–ciphertext pairs. The operating principle of this attack is based on computing all possible encryptions of the plaintext using the first of the two keys, followed by decrypting the known result (the ciphertext) with every possible second key. The set of results from the first process is then compared with the set of results from the second process. If a match can be found, there is a certain probability that the two keys have been discovered. The level of confidence that the correct key has been found can be increased by making the same comparisons using additional known plaintext–ciphertext pairs. As can be seen, the amount of effort required for this attack is not significantly greater than the amount of effort needed for a normal attack that requires the entire key space to be searched. Consequently, cascaded double encryption is not used with the DES algorithm.

The process that is used instead is called triple-DES. In this mode, three sequential CBC-mode DES operations are performed using alternating encryption and decryption. Blocks that have been encrypted in this manner are decrypted by reversing the order of the operations (in other words: decryption, encryption and then decryption). If all three keys are the same, the result of the alternating encryption and decryption operations is the same as that obtained by a single encryption. This is the reason for not using a sequence of three encryption operations.

If the three DES operations using three keys are applied directly to each plaintext block in turn, the process is referred to as ‘DES in the inner CBC mode’. If instead the plaintext is first completely encrypted using the first key and the result is then further encrypted in a similar manner, the process is referred to as ‘DES in the outer CBC mode’. The outer CBC mode is more resistant to attack and is therefore generally recommended [Schneier 96].

The triple-DES algorithm has several other names, such as TDES, DES-3, 3-DES and 2-DES. Actually, the terms ‘triple-DES’ and ‘3-DES’ only mean that three 56-bit keys are used. If the first and third keys are the same, this is called 2-DES, but if the three keys are all different, the designation 3-DES is often used. The key length must always be stated with triple DES in order to unambiguously specify the algorithm.

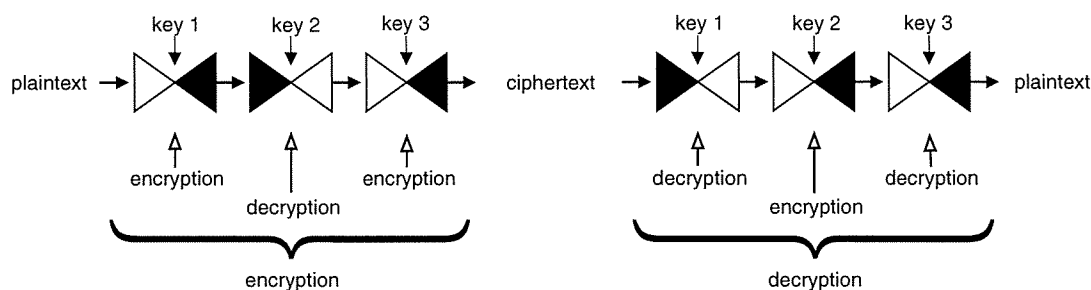


Figure 4.29 Operating principle of data encryption using triple-DES in the outer CBC mode. Key 1 is normally chosen to be the same as key 3, so the effective key length is 2×56 bits (112 bits). Less commonly, three independent keys are used, yielding a key length of 3×56 bits (168 bits)

The triple-DES algorithm is significantly more secure than sequential double encryption using two different keys, since the meet-in-the-middle attack is not effective against this method. Three 56-bit keys are needed instead of only one, but the first and third keys are usually the same. This yields a key length of 2×56 bits. This means that this procedure is data-compatible with the normal DES algorithm and that it imposes no additional costs except for the doubled key size. This in particular is one of the main reasons for the widespread use of triple-DES in smart cards. It is primarily used for deriving keys and protecting very sensitive data, such as when transferring keys, due to its improved level of security compared with single encryption.

4.7.2 Asymmetric cryptographic algorithms

In 1976, Whitfield Diffie and Martin E. Hellman described the idea of developing an encryption algorithm based on two different keys [Diffie 76]. One of these keys was to be public, the other secret or 'private'. This would allow persons to encrypt a message using the public key, with only the owner of the private key being able to decrypt it. This in turn would eliminate the problems associated with exchanging and distributing secret symmetric keys. In addition, it would for the first time make certain other processes possible, such as generating digital signatures that can be verified by everyone.

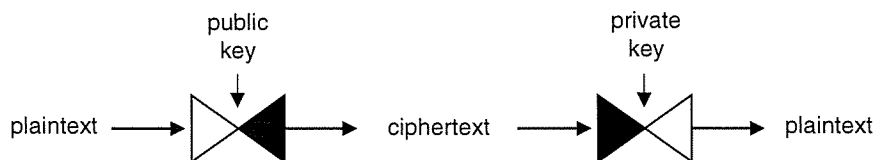


Figure 4.30 Encryption and decryption using a public-key algorithm

The RSA algorithm

Two years later, Ronald L. Rivest, Adi Shamir and Leonard Adleman presented an algorithm that met the above-mentioned criteria [Rivest 78]. This algorithm, which is called the RSA

Table 4.10 Summary of symmetric and asymmetric cryptographic algorithms used in smart cards, with their input and output parameters

Name	Type	Plaintext length	Ciphertext length	Key length
DES	Symmetric	8 bytes	8 bytes	56 bits
Triple DES (with two keys)	Symmetric	8 bytes	8 bytes	2 × 56 bits (112 bits)
Triple DES (with three keys)	Symmetric	8 bytes	8 bytes	3 × 56 bits (168 bits)
IDEA	Symmetric	8 bytes	8 bytes	128 bits
AES	Symmetric	8 bytes	16 bytes	128 bits (16 bytes) 192 bits (24 bytes) 256 bits (32 bytes)
RSA	Asymmetric	512 bits (64 bytes) 768 bits (96 bytes) 1024 bits (128 bytes) 2048 bits (256 bytes) 4096 bits (512 bytes)	(512 bytes) (64 bytes) 768 bits (96 bytes) 1024 bits (128 bytes) 2048 bits (256 bytes) 4096 bits (512 bytes)	512 bits (64 bytes) 768 bits (96 bytes) 1024 bits (128 bytes) 2048 bits (256 bytes) 4096 bits (512 bytes)
DSS (512 bits)	Asymmetric	20 bytes	20 bytes	(64 + 20) bytes

algorithm after the initials of its inventors, is the best-known and most versatile asymmetric encryption algorithm presently in use. Its very simple operating principle is based on the arithmetic of large integers. The two keys are generated from two large prime numbers.⁶

The encryption and decryption processes can be expressed mathematically as follows:

$$\text{encryption: } y = x^e \bmod n$$

$$\text{decryption: } x = y^d \bmod n$$

where x = plaintext

y = ciphertext

e = public key

⁶ Whitfield Diffie and Martin E. Hellman discovered the principle of public-key algorithms, while Ronald L. Rivest, Adi Shamir and Leonard Adleman produced the first implementation. However, public-key algorithms had already been discovered several years earlier in the field of military cryptography. In the period 1969–1973, James Ellis, Clifford Cocks and Malcolm Williamson, working at the British Government Communication Headquarters (GCHQ), developed both the principle of public-key algorithms and the asymmetric encryption algorithm now known as the RSA algorithm, several years before they were discovered by civilians. Unfortunately, these cryptologists were pledged to secrecy and were not allowed to publish anything about their work until after 1997 [Levy 99]

$$\begin{aligned}
 d &= \text{private key} \\
 n &= \text{public modulus} = p \cdot q \\
 p, q &= \text{secret prime numbers}
 \end{aligned}$$

Before being encoded, the plaintext block must be padded to the appropriate block size, which varies in the RSA algorithm according to the length of the key used. Encryption itself is performed by exponentiation of the plaintext followed by a modulo operation. The result of this process is the ciphertext. This can only be decoded if the private key is known. The decryption process is analogous to the encryption process.

The security of the algorithm is thus based on the difficulty of factoring large numbers. It is quite easy to compute the public modulus from the two prime numbers by multiplication, but it is very difficult to decompose the modulus into its two prime factors, since there is no effective algorithm for this operation.

The RAM capacity of smart cards is not sufficient for performing exponential operations on large numbers as required for encryption and decryption, since the numbers become very large before being subjected to the modulo operation. For this reason, modular exponentiation is used, which means that the intermediate result of the calculation never exceeds the value of the modulus. For example, if the value of $x^2 \bmod n$ must be calculated, the expression $(x \cdot x) \bmod n$ is not evaluated directly, since the intermediate result $(x \cdot x)$ would be excessively large before being reduced by the modulo operation. Instead, the expression $((x \bmod n) \cdot (x \bmod n)) \bmod n$ is evaluated, which yields the same mathematical result. The advantage of this is that it requires significantly fewer calculations and less memory, since the intermediate results are immediately reduced in size.

An additional way to increase the speed of the RSA algorithm is to use the Chinese remainder theorem for the calculations.⁷ Of course, a prerequisite for using the Chinese remainder theorem is that both of the secret prime numbers p and q are known, which means that it can only be used for decryption (which means for signing).

The private key should be as long as possible, since this impedes attempts to break the code. Public and private keys may have different lengths, and in fact this is usually the case, since the time required to verify a digital signature can be considerably reduced by making the public key as short as possible. The fourth Fermat number is frequently used as a public key. This prime number has the value of $2^{16} + 1 = 65537$, and due to its small size it is very well suited to quickly verifying digital signatures. The numbers 7 and 17 are likewise used.

Table 4.11 Typical public keys for the RSA algorithm

Public key	Remarks
$2 = 2^1$	The only even prime number; used for the Rabin procedure
$3 = (2^1 + 1) = 2^2 - 1$	The smallest odd prime number
$17 = (2^4 + 1) = 2^8 - 1$	—
$65537 = (2^{16} + 1) = 2^{32} - 1$	The fourth Fermat number F_4

⁷ For additional information regarding both procedures, see [Simmons 92, Schneier 96]

If an attacker succeeds in factoring the public modulus into its two prime components, he could then reproduce the entire encryption process. With a small value, such as 33, it is easy to factor the modulus, but there is presently no fast algorithm that can be used to factor large numbers. If the values of both prime factors can be found, the system is thereby broken, since the private key is then known.⁸

Consequently, a requirement for RSA keys is that they are sufficiently long. A length of 512 bits (64 bytes) is presently considered to be the lower limit. In any case, keys with a length of 768 bits (96 bytes) and 1024 bits (128 bytes) are presently used. In the coming years, the first 2048-bit (256-byte) key will come into use. The amount of computational effort needed for encryption and decryption increases with the key length. This increase is not linear, but instead approximately exponential.

Smart card microcontrollers, with their 8-bit CPUs, are normally not capable of performing an RSA computation in less than a few minutes. However, there are now microcontrollers with arithmetic coprocessors that have been specially developed for fast exponentiation. With such coprocessors, it is possible to perform RSA computations in an acceptable length of time with reasonable software overhead. The size of the code for a hardware-supported 512-bit RSA algorithm is around 300 bytes. Around 1 kB of assembler code in the smart card is needed for 768-bit and 1024-bit keys. As can be seen from Table 4.12, even with a 512-bit key the number of possible prime numbers is so large that collisions between two different key pairs will never occur.⁹

Table 4.12 Typical RSA key lengths with characteristic parameters. The ratio NS/PN indicates the relationship between the number of non-prime numbers and the number of prime numbers. The reciprocal of this is the probability that a random number in the number space is a prime. This is very important with regard to the length of time that is required to generate an RSA key

Key length	Maximum number of digits	Size of the number space (NS)	Number of prime numbers in the number space (PN)	NS/PN
8 bits (1 byte)	3	256	54	≈ 4.7
40 bits (5 bytes)	13	$\approx 1.1 \times 10^{12}$	$\approx 3.9 \times 10^{10}$	≈ 28
512 bits (64 bytes)	155	$\approx 1.3 \times 10^{154}$	$\approx 3.8 \times 10^{151}$	≈ 342
768 bits (96 bytes)	232	$\approx 1.6 \times 10^{231}$	$\approx 2.9 \times 10^{228}$	≈ 552
1024 bits (128 bytes)	309	$\approx 1.8 \times 10^{308}$	$\approx 2.5 \times 10^{305}$	≈ 720
2048 bits (256 bytes)	617	$\approx 3.2 \times 10^{616}$	$\approx 2.3 \times 10^{613}$	≈ 1391
4096 bits (512 bytes)	1234	$\approx 1.0 \times 10^{1233}$	$\approx 2.5 \times 10^{1229}$	≈ 4000

However, one of the strengths of the RSA algorithm is that it is not limited to a particular key length, in contrast to algorithms such as DES. If increased security is needed, longer keys

⁸ As early as the summer of 1994, a 426-bit key was broken in eight months using approximately 1600 computers connected via the Internet. The total amount of computation amounted to 5000 MIPS-years. A 100-MHz Pentium processor, for comparison, has a capacity of 50 MIPS. In 19991, a 512-bit key was successfully factored in seven months using 292 networked computers

⁹ The largest number that can be represented with 512 bits is $2^{512} - 1$, or in full: 13,407,807,929,942,597, 099,574,024,998,205,846,127,479,365,820,592,393,377,723,561,443,721,764,030,073,546,976,801,874,298,166, 903,427,690,031,858,186,486,050,853,753,882,811,946,569,946,433,649,006,084,095

can be used without modifying the algorithm. The RSA algorithm is thus scaleable. However, computation time and amount of memory space needed must be kept in mind, since even 768-bit keys are presently still considered to be secure. With current factoring algorithms, a good rule of thumb is that increasing the key length by 15 bits doubles the effort of computing the factors.¹⁰ Andrew Odlyzko [Odlyzko 95] provides an excellent summary of the internationally available and required processing capacity for factoring integers.

Although the RSA algorithm is very secure, it is rarely used to encrypt data, due to its long computation time. It is primarily used in the realm of digital signatures, where the benefits of an asymmetric procedure can be fully realized. The greatest drawback of the RSA algorithm with regard to smart cards is the amount of memory space required for the key. The complexity of the key generation process also causes problems in certain cases.

Widespread use of the RSA algorithm is restricted by patent claims that have been made in several countries and by major import and export restrictions imposed on equipment that employs this algorithm. Smart cards with RSA coprocessors fall under these restrictions, which considerably hinders their use internationally.

Table 4.13 Sample computation times for RSA encryption and decryption as a function of key length. The indicated values are in part subject to considerable variation, since they are strongly dependent on the microcomputer used, the bit structure of the key and the use of the Chinese remainder algorithm (which can only be used for signing)

Implementation	Mode	512 bits	768 bits	1024 bits	2048 bits
Smart card without NPU, 8-bit CPU, 3.5 MHz clock	Signing	20 min	—	—	—
Smart card without NPU, 8-bit CPU, 3.5 MHz clock (with Chinese remainder theorem)	Signing	6 min	—	—	—
Smart card with NPU, 3.5 MHz clock	Signing	308 ms	910 ms	2.0 s	—
Smart card with NPU, 3.5 MHz clock (with Chinese remainder theorem)	Signing	84 ms	259 ms	560 ms	—
Smart card with NPU, 4.9 MHz clock	Signing	220 ms	650 ms	1.4 s	—
Smart card with NPU, 3.5 MHz clock	Verifying	—	—	1.04 s	—
Smart card with NPU, 4.9 MHz clock (with Chinese remainder theorem)	Signing	60 ms	185 ms	400 ms	—
Smart card with NPU and PLL	Verifying	60 ms	185 ms	400 ms	—
PC (Pentium, 200 MHz)	Signing	12 ms	46 ms	60 ms	—
PC (Pentium, 200 MHz)	Verifying	2 ms	4 ms	6 ms	—
RSA integrated circuit	Signing	1.6 ms	—	—	—

¹⁰ As of January 1998, the largest known prime number had 909,256 digits and a value of $2^{3,402,377} - 1$

Generating RSA keys

Keys for the RSA algorithm are generated using a simple process. The following is a small worked-through example:

1. First, select two prime numbers p and q : $p = 3; q = 11$
2. Next, calculate the public modulus: $n = p \cdot q = 33$
3. Calculate the temporary variable z for use during key generation: $z = (p - 1) \cdot (q - 1)$
4. Calculate a public key e which satisfies the conditions $e < z$ and $\text{gcd}(z, e) = 1$ (that is, the greatest common denominator of z and e is 1). Since there are several numbers that meet these conditions, select one of them: $e = 7$
5. Calculate a private key d that satisfies the condition $(d \cdot e) \bmod z = 1$: $d = 3$

This completes the computation of the keys. The public and private keys can now be tested for encryption and decryption using the RSA algorithm, as illustrated in the following numeric example:

1. Use the number '4' as the plaintext x ($x < n$): $x = 4$
2. Encrypt the text: $y = 4^7 \bmod 33 = 16$
2. The result of the calculation is the ciphertext y : $y = 16$
3. Decrypt the ciphertext: $x = 16^3 \bmod 33 = 4$

The result of decrypting the ciphertext is again the original plaintext, as expected.

In actual practice, key generation is more laborious, since it is very difficult to test large numbers to determine if they are prime. The well-known sieve of Eratosthenes cannot be used here, since it requires prior knowledge of all prime numbers smaller than the number being tested. This is practically impossible for numbers as large as 512 bits. Consequently, probabilistic tests are used to determine the likelihood that the selected number is a prime number. The Miller–Rabin test and the Solovay–Strassen test¹¹ are typical examples of such tests. To avoid having to use these time-consuming tests more than necessary, randomly generated candidate numbers are first tested to see if they have any small prime factors. If the randomly generated number can be exactly divided by a small prime number, such as 2, 3, 5 or 7, it obviously cannot be a prime number. Once it has been determined that the number to be tested does not have any small prime factors, a prime number test such as the Miller–Rabin test can be used. The principle of this test is illustrated in Figure 4.31 and described in detail in the appendix of the IEEE 1363 standard.¹²

¹¹ The procedure and the algorithm are described by Alfred Menezes [Menezes 97]

¹² Many tips and criteria that must be taken into account for the generation of prime numbers can be found in an article by Robert Silverman [Silverman 97]

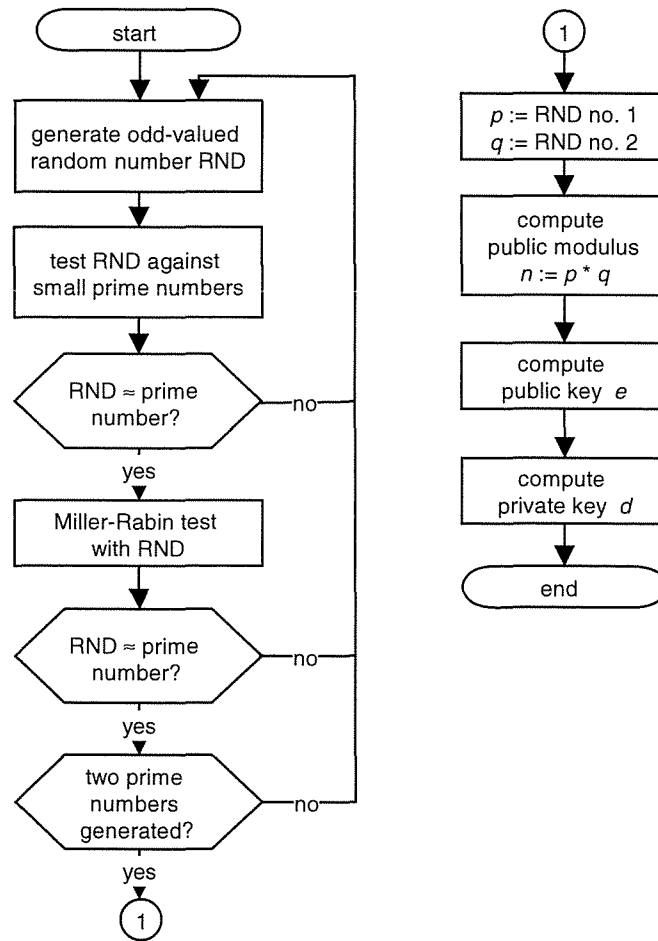


Figure 4.31 Basic procedure for generating RSA keys for use in smart cards

The algorithms for generating RSA keys have a special feature, which is that the time required to generate a key pair (a public key together with a private key) is only statistically predictable. This means that it is only possible to say that there is a certain probability that key generation will take a given amount of time. A definitive statement such as '... will take x seconds' is not possible, due to the need to run the prime number test on the random number. The time required to perform this test is not deterministically predictable.

The DSS algorithm

In mid-1991, the NIST (US National Institute of Standards and Technology) published the design of a cryptographic algorithm for adding signatures to messages. This algorithm, which has since been standardized in the US (FIPS 186), has been named the Digital Signature Algorithm (DSA), and the standard that describes it is called the Digital Signature Standard (DSS). The DSA and RSA algorithms are the two most widely used procedures for generating digital signatures. The DSA algorithm is a modification of the El Gamal procedure. The background for the standardization of this algorithm is that a procedure was wanted that could be used to generate signatures but not to encrypt data. For this reason, the DSA algorithm is

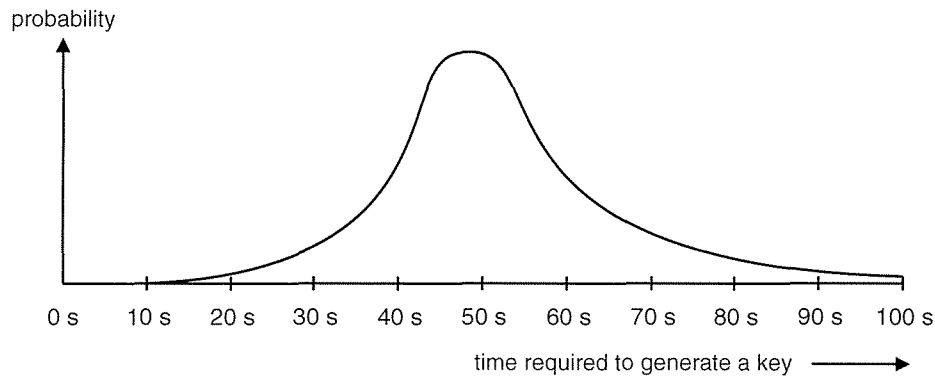


Figure 4.32 Typical time behavior of a probabilistic algorithm for generating key pairs for the RSA algorithm. The vertical axis shows the probability that a given amount of time will be required to generate a 1024-bit key in a smart card. Consequently, key generation takes 50 s on average in this example. The total area under the curve has a probability of 1

Table 4.14 Examples of the time required to generate a pair of public and private keys for the asymmetric RSA cryptographic algorithm. Exact times cannot be given, since the duration of the key generation process depends on whether the generated random numbers are prime, among other things

Generating a public/private key pair for the RSA algorithm	Typical time	Possible time
Smart card, 512-bit key, 3.5 MHz	6 s	≈ 1 s to ≈ 20 s
Smart card, 1024-bit key, 3.5 MHz	14 s	≈ 6 s to ≈ 40 s
Smart card, 2048-bit key, 3.5 MHz	80 s	≈ 6 s to ≈ 40 s
PC (Pentium, 200 MHz), 512-bit key	0.5 s	—
PC (Pentium, 200 MHz), 1024-bit key	2 s	—
PC (Pentium, 200 MHz), 2048-bit key	36 s	—

more complicated than the RSA algorithm. However, it has been shown that it is possible to encrypt data using this algorithm [Simmons 98].

In contrast to the RSA algorithm, the security of the DSS algorithm does not depend on the problem of factoring large numbers, but rather on the discrete logarithm problem. The expression $y = a^x \bmod p$ can be computed quickly, even with large numbers. However, the reverse process, which is calculating the value of x for given values of y , a and p , requires a very large amount of computational effort.

With all signature algorithms, the message to be signed must first be reduced to a predefined length using a hash algorithm. The NIST therefore published a suitable algorithm for use with the DSS algorithm. This is named SHA-1 (Secure Hash Algorithm).¹³ This variant of the MD5 hash algorithm generates a 160-bit hash value from a message of any arbitrary length. Computations for the DSS algorithm, like those for the RSA algorithm, are performed using only integers.

¹³ See Section 4.9, 'Hash Functions'

To compute a signature with the DSA algorithm, the following global values must first be determined:

p (public): prime number with a length of 512 to 1024 bits, evenly divisible by 64
 q (public): 160-bit prime factor of $(p - 1)$
 g (public): $g = h^{(p-1)/q}$
 where h is an integer satisfying the conditions $h < p - 1$ and $g > 1$

The private key x must satisfy the following condition:

$$x < q$$

The public key y is computed as follows:

$$y = g^x \bmod p$$

Once all of the necessary keys and numbers have been determined, the message m can be signed as follows:

Generate a random number k , where $k < q$: k
 Compute the hash value of m : $H(m)$
 Calculate r : $r = (g^k \bmod p) \bmod q$
 Calculate s : $s = k^{-1} (H(m) + x \cdot r) \bmod q$

The two values r and s are the digital signature of the message. With the DSS algorithm, the signature consists of two numbers, instead of only one number as with the RSA algorithm.

The signature is verified as follows:

Calculate w : $w = s^{-1} \bmod q$
 Calculate $u1$: $u1 = (H(m) \cdot w) \bmod q$
 Calculate $u2$: $u2 = (r \cdot w) \bmod q$
 Calculate v : $v = ((g^{u1} \cdot y^{u2}) \bmod p) \bmod q$

If the condition $v = s$ is satisfied, the message m has not been altered and the digital signature is authentic.

In practice, the RSA algorithm has achieved more widespread use than the DSS algorithm, which up to now has seen only very limited use. The original idea of standardizing a signature algorithm that cannot be used for encryption, which led to the DSS algorithm, has largely come to nothing. The complexity of this algorithm also discourages its widespread use. Nonetheless, for many institutions the fact that the standard exists and the political pressure to generate signatures using the DSS and SHS represent strong arguments in its favor.

Table 4.15 Examples of computation times for the DSA algorithm as a function of the clock rate, divided into the times required for verifying (encrypting) and generating (decrypting) a signature. These values are subject to considerable variation, since they depend strongly on the bit structure of the key. The computation time can be reduced by precomputation

Implementation	Verifying a 512-bit signature	Generating a 512-bit signature
Smart card with 3.5-MHz clock	130 ms	70 ms
Smart card with 4.9-MHz clock	90 ms	50 ms
PC (80386, 33 MHz)	16 s	35 ms

Using elliptic curves as asymmetric cryptographic algorithms

In addition to the two well-known asymmetric cryptographic algorithms, RSA and DSA, there is a third type of cryptography that is used for digital signatures and key exchanges in the realm of smart cards. It is based on elliptic curves (EC).

In 1985, Victor Miller and Neal Koblitz independently proposed the use of elliptic curves for constructing asymmetric cryptographic algorithms. The properties of elliptic curves are well suited to such applications, and in the course of the following years, practical cryptographic systems based on these proposals were developed. In general, they are usually referred to as elliptic curve cryptosystems (ECC).

Elliptic curves are sets of smooth curves that satisfy the equation $y^2 = x^3 + ax + b$ within a finite three-dimensional space. No point is allowed to be a singularity. This means, for instance, that $4a^2 + 27b^2 \neq 0$. In the realm of cryptography, the finite spaces $\text{GF}(p)$, $\text{GF}(2^n)$ and $\text{GF}(p^n)$ are used, where p is a prime number and n is a positive integer greater than 1.

The mathematics of cryptographic systems based on elliptic curves are relatively difficult. For this reason, you are referred to the book by Alfred Menezes on the subject [Menezes 93]. The very comprehensive IEEE 1363 public-key cryptography standard and the ISO/IEC 15946 series of standards dealing with elliptic curves also provide good synopses of elliptic curves and other asymmetric cryptographic techniques.

The major advantages of asymmetric cryptographic systems based on elliptic curves are that they require much less computational capacity than systems such as RSA (for instance), and that the same level of cryptographic strength can be attained with significantly shorter keys. For example, roughly the same amount of computation is required to break an ECC algorithm with a 160-bit key as an RSA algorithm with a 1024-bit key. Similarly, an ECC algorithm with a 256-bit key corresponds to an RSA algorithm with a 2048-bit key, while an ECC algorithm with a 320-bit key roughly corresponds to an RSA algorithm with a 5120-bit key. This cryptographic strength and the relatively small size of the keys are precisely the reasons why ECC systems are found in the realm of smart cards.

The arithmetic processing components of modern-day smart card microcontrollers generally support ECC, which means that a relatively high computational speed is available. As with the RSA algorithm, the key length is an important characteristic of these asymmetric cryptographic algorithms.

Interestingly enough, cryptographic systems based on elliptic curves require so little processing capacity that they can even be implemented in microcontrollers lacking coprocessors.

Some typical times for generating and verifying signatures are shown in Table 4.16. An 8-bit microcontroller clocked at 3.5 MHz without a coprocessor requires approximately one second to generate a 160-bit ECC key pair using a look-up table approximately 10 kB in size. This time can be reduced to 200 ns using a coprocessor.

Table 4.16 Sample processing times for cryptographic algorithms based on elliptic curves in $GF(p)$. The remarkably good times for smart cards without coprocessors are achieved using table look-up to accelerate certain time-intensive computations (table size approximately 10 kB)

Implementation	Generating a 135-bit signature	Verifying a 135-bit signature
Smart card, 3.5-MHz clock and 8-bit processor	1 s	4 s
Smart card, 3.5-MHz clock and numeric coprocessor	150 ms	450 ms
PC (Pentium III, 500 MHz)	10 ms	20 ms

One factor limiting the use of elliptic curves for asymmetric cryptographic algorithms is that they are regarded as a relatively new discovery in the cryptographic world, even though they have been known for a long time. It will no doubt take some time until the use of ECC systems becomes commonplace in the cautious world of cryptographers and smart card application designers, despite the fact that cryptographic systems based on elliptic curves presently offer the highest level of security per bit relative to all other asymmetric methods.

4.7.3 Padding

In smart cards, the DES algorithm is primarily used in the two block-oriented modes (ECB and CBC). However, since the data communicated to the card do not always fit exactly into a certain number of blocks, it is occasionally necessary to fill up a block. Filling up a data block so that its length is an exact multiple of a given block size is called *padding*.

The recipient of a padded data block has a problem after the data have been decrypted, since he does not know where the actual data stop and the padding bytes start. One solution to this would be to state the length of the message at the beginning of the message, but this would change the structure of the message, which is generally undesirable. It would also be especially onerous with data that do not always have to be encrypted, since in this case no padding would be needed and thus no length as well. In many cases, therefore, the structure of the message may not be changed.

This means that a different method must be used to identify the padding bytes. The algorithm defined in the ISO/IEC 9797 standard is described here in detail as an example, although there are a variety of other methods available. The most significant bit (msb) of the first padding byte following the useful data is set to 1. This byte thus has the hexadecimal value '80'. If additional padding bytes are needed, they have the value '00'. The recipient of the padded message thus searches from the beginning to the end of the message for a byte with the msb set to 1, or for the value '80'. If such a byte is found, the recipient knows that this byte and all subsequent bytes are padding bytes and not part of the message.

In this regard, it is important for the recipient to know whether messages are always padded or padded only if necessary. If padding only takes place when the length of the data to be

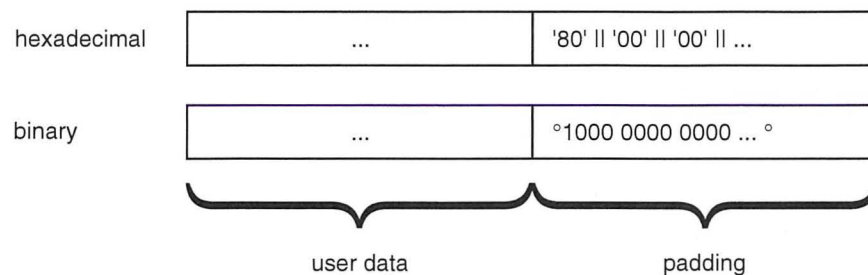


Figure 4.33 Data padding according to ISO/IEC 9797, Method 2

encrypted is not an integer multiple of the block length, the recipient must take this into account. Consequently, there is often an implicit understanding that padding always takes place, which of course has the disadvantage that occasionally an unnecessary block of padding data must be encrypted, transferred and decrypted.

In some applications, only the value '00' is used for padding. This is because this value is normally used for padding in MAC computations, and using only one padding algorithm reduces the size of the program code. Of course, in this case the application must know the exact structure of the data to allow it to distinguish between user data and padding.

Table 4.17 Typical padding methods using in the smart card realm. The data to be padded are designated as 'data'

Padding format	Description
ISO/IEC 9797	<p>This padding format is used for generating MACs and for encryption</p> <p>Method 1: the data to be padded are padded using '00'</p> <p>Formal representation: data $n \times '00'$</p> <p>Method 2: '80' is appended to the data to be padded, which are then padded using '00'</p> <p>Formal representation: data '80' $n \times '00'$</p>
ISO/IEC 9796-2	<p>This padding method is used for digital signatures. The data to be padded are appended to a bit sequence starting with $^{\circ}11^{\circ}$ and ending with $^{\circ}1^{\circ}$, with a number of $^{\circ}0^{\circ}$ characters in between as needed for padding, and the tag 'BC' is appended to the data. In addition, a random number can be integrated into the padding sequence in order to individualize the data to be padded</p> <p>Formal representation with bitwise padding: '60' $n \times '00'$ '01' data 'BC'</p> <p>Formal representation with bitwise padding and individualized data: '60' $n \times '00'$ '01' RND data 'BC'</p>
PKCS #1	<p>The Type 1 version of this padding format is used for digital signatures, while the Type 2 version is used for generating MACs and encryption. The data to be padded are preceded by a tag and a fixed value or random number having the length necessary for the padding</p> <p>Formal representation, Type 1: '00' '01' $n \times 'FF'$ '00' data</p> <p>Formal representation, Type 2: '00' '02' $n \times \text{RND}$ '00' data</p>

4.7.4 Message authentication code and cryptographic checksum

The authenticity of a message is far more important than its confidentiality. The term ‘authenticity’ means that the message has not been altered or manipulated, and is thus genuine. To ensure authenticity, a ‘message authentication code’ (MAC) is computed and appended to the message before it is sent to the recipient. The recipient can then compute the MAC for the message and compare it with the received MAC. If the two values match, the message has not been altered during its journey.

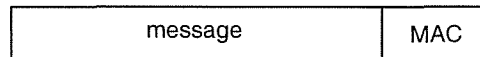


Figure 4.34 The usual arrangement of the message and the message authentication code (MAC)

A cryptographic algorithm with a secret key is used to generate a MAC. This key must be known to both parties to the communication. In principle, a MAC is a sort of error detection code (EDC), which can naturally only be verified if the associated secret key is known. For this reason, the term ‘cryptographic checksum’ (CCS) is also used (as well as some other terms), but technically a CCS is fully identical to a MAC. In general, the difference between the two terms is that ‘MAC’ is used for data transmission and ‘CCS’ is used for all other applications. The term ‘signature’ is often encountered as an equivalent to ‘MAC’. However, this is not the same as a ‘digital signature’, since the latter is generated using an asymmetric cryptographic algorithm.

In principle, any cryptographic algorithm can be used to compute a MAC. In practice, however, the DES algorithm is used almost exclusively. This algorithm is used here to demonstrate the process (see Figure 4.35).

If the message is encrypted using the DES algorithm in the CBC mode, each block is linked to its previous block. This means that the final block depends on all previous blocks. This final block, or a portion of it, represents the MAC of the message. However, the actual message remains in plaintext, rather than being transmitted in encrypted form.

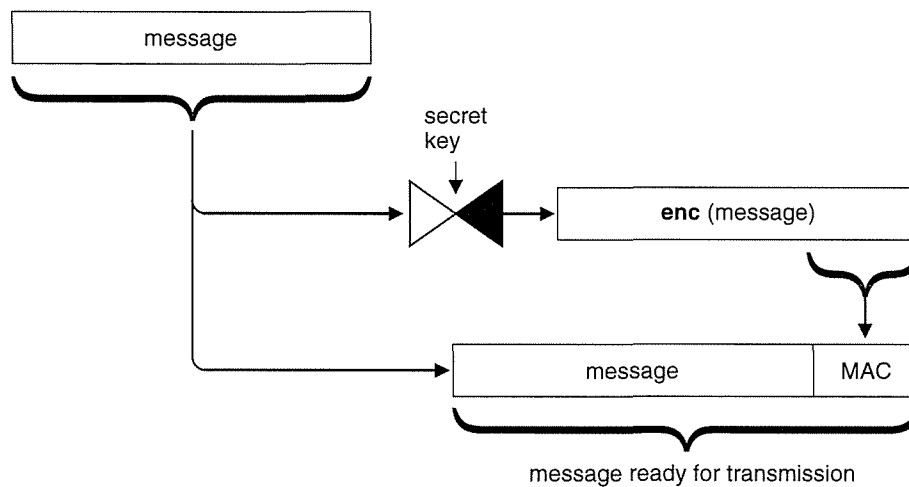


Figure 4.35 Example of a MAC computation process

There are a few important conditions relating to generating a MAC using the DES algorithm. If the length of the message is not an exact multiple of eight bytes, it must always be extended, which generally involves padding. However, in most cases only the value '00' is used for padding (in line with ANSI X.99 – Message Authentication). This is allowed in this case because there must be prior agreement regarding the length and location of the MAC within the message. The actual MAC consists of the left-most (most significant) four bytes of the final block produced by CBC-mode encryption. However, the padding bytes are not sent when the message is transmitted. This limits the data to be transmitted to the protected data and the appended MAC.

4.8 KEY MANAGEMENT

The sole objective of all administrative principles relating to keys for cryptographic algorithms is to minimize the consequences to the system and the smart card application if one or more secret keys become known to unauthorized persons. If it could be guaranteed that the keys would always remain secret, a single secret key for all smart cards would be sufficient. However, it is impossible to guarantee such secrecy.

Using the security-enhancing principles described here for keys used with cryptographic algorithms causes the number of keys to increase dramatically. If all of the principles and methods described in this section are implemented in a single smart card, the keys will usually take up more than half of the memory available for application data.

However, it is not always necessary to use every possible principle and method, depending on the application. For example, there is no need to support multiple generations of keys if the card is valid for only a limited length of time, since the additional administrative effort and memory space cannot be justified.

4.8.1 Derived keys

Since smart cards, in contrast to terminals, can be taken home by anyone and possibly subjected to thorough and painstaking analysis, they are naturally exposed to the most severe attacks. If no master key is present in the card, the consequences of a successful attempt to read out the card contents can be minimized. Consequently, the keys that are found in the card are only those that have been derived from a master key.

Derived keys are generated using a cryptographic algorithm. The input values are a card-specific feature and a master key. The triple-DES or AES algorithm is usually used. For the sake of simplicity, the card number is usually used as the specific feature. This number, which is generated when the card is manufactured, is unique in the entire system and can be used throughout the system to identify the card.

Derived keys are thus unique. One function that can be used to generate derived keys, as illustrated in Figure 4.36, is:

$$\text{derived key} = \text{enc}(\text{master key}; \text{card number})$$

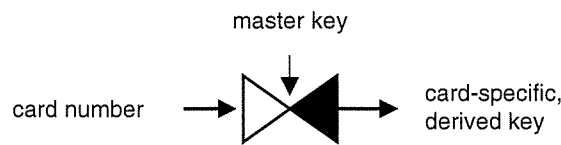


Figure 4.36 A possible method for generating a derived, card-specific symmetric key from the card number and a master key

4.8.2 Key diversification

In order to minimize the consequences of a key being compromised, a separate key is often used for each cryptographic algorithm. For example, different keys can be used for signatures, secure data transmission, authentication and data encrypting. For each type of key, there must be a separate master key from which the individual keys can be derived.

4.8.3 Key versions

It is normally not adequate to employ only one key generation for the full lifetime of a smart card. For example, suppose that a master key could be computed as the result of a successful attack. In this case, all application vendors would have to shut down their systems and card issuers would have to replace all their cards. The resulting loss would be enormous. Consequently, all modern systems include the possibility of switching to a new key generation.

Switching to a new generation of keys may be forced by the fact that a key has been compromised, but it can also take place routinely at a fixed or variable interval. The result of a switch is that all of the keys in the system are replaced by new ones, without any need for the cards to be recalled. Since the master keys are located in the terminals and the higher level parts of the system, a secure data exchange is all that is needed to provide new, confidential keys to the terminals.

4.8.4 Dynamic keys

In many applications, and in particular in the area of data transmission security, it is common practice to use dynamic keys. Such keys are also called 'temporary keys' or 'session keys'. To generate a dynamic key, one of the two communicating parties first generates a random number, or some other value for use in a specific session, and passes it to the other party. The further course of the process depends on whether cryptographic algorithms used are only symmetric or also asymmetric.

Dynamic keys with symmetric cryptographic algorithms

For procedures that use only symmetric cryptographic algorithms, the random number generated by one of the two parties is sent as plaintext to the other party. The smart card and the

terminal then encrypt this number using a derived key. The result, as shown in Figure 4.37, is a key that is valid only for one particular session.

$$\text{dynamic key} = \text{enc}(\text{derived key}; \text{random number})$$

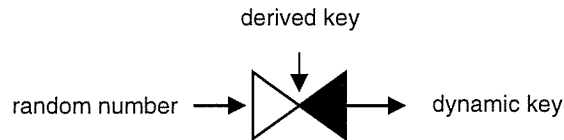


Figure 4.37 A possible way to generate a dynamic key using a random number and a derived key

The main advantage of dynamic keys is that they are different for each session, which makes attacks significantly more difficult. However, care must be taken when a dynamic key is used to generate a signature, since the dynamic key will also be needed to verify the signature. This key can only be generated using the same random number as was used when the signature was created. This means that whenever a dynamic key is used for a signature, the random number used to generate the key must be retained for use in verification, which means it must be stored.

The ANSI X 9.17 standard proposes a different method for generating derived and dynamic keys. Although it is somewhat more complicated than the previously described method, it is widely used in financial transaction systems. This method requires two inputs: a value T_i that is independent of the time or session and a key Key_{Gen} that is reserved for generating new keys. The resulting initial key Key_i can be used to compute as many additional keys as desired. This key generation method has the additional advantage that it cannot be computed in reverse; in other words, it is a one-way function:

$$Key_{i+1} = \text{enc}(Key_{Gen}; \text{enc}(Key_{Gen}; (T_i \text{ XOR } Key_i)))$$

Exchanging dynamic keys using an asymmetric cryptographic algorithm

Figures 4.38 and 4.39 show procedures for generating and subsequently exchanging a symmetric dynamic key for message encryption. An asymmetric cryptographic algorithm, such as RSA or DES, is used for key exchange. A similar process is used in PGP, for example, which uses the IDEA and RSA algorithms. The basic advantage of this hybrid process is that the actual encryption of large volumes of data can be performed using a symmetric cryptographic algorithm, which has significantly higher throughput than an asymmetric algorithm.

4.8.5 Key parameters

A mechanism that is as simple as possible is needed to allow the key stored in the card to be externally addressed. The smart card operating system must also always ensure that the key can only be used for its intended purpose. For instance, it must prevent an authentication key from being used for encrypting data. Besides the intended use, the key number must be known

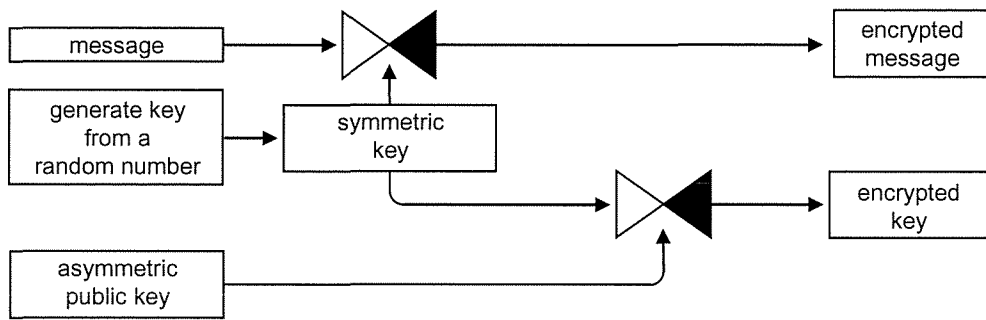


Figure 4.38 Sample procedure for key exchange using a combination of symmetric and asymmetric cryptographic algorithms. An encrypted dynamic symmetric key is first generated and then exchanged between two parties using an asymmetric cryptographic algorithm. The generation and exchange of the key pair for the asymmetric cryptographic algorithm, which takes place separately and in advance, is not shown

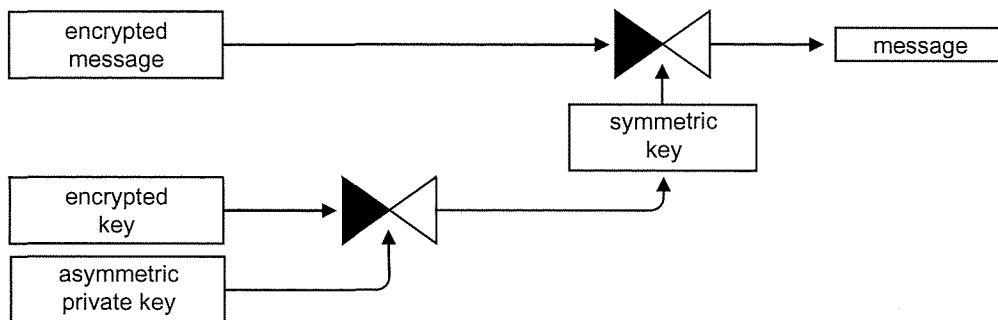


Figure 4.39 Sample procedure for key exchange using a combination of symmetric and asymmetric cryptographic algorithms. A previously encrypted dynamic symmetric key is recovered using an asymmetric cryptographic algorithm. The generation and exchange of the key pair for the asymmetric cryptographic algorithm, which takes place separately and in advance, is not shown

for it to be addressed. This number is the actual reference to the key. In addition, the version number is also needed to address a specific key.

Some smart card operating systems cause a retry counter associated with the key to be incremented each time a failure occurs in some activity that uses the key, such as an authentication. This can be used to quite reliably prevent the key value from being fished out by repeated trials, although this type of an attack does not represent a serious risk due to the long processing times in the card. If the retry count reaches its maximum value, the key is blocked and cannot be further used. The retry counter is reset to zero if the attempt to use the key is successful. Such a mechanism must always be used with great care, since an incorrect master key in a terminal could easily lead to massive card failures. A retry counter can normally only be reset using a special terminal, and the identity of the cardholder must be verified before this is done.

Some systems prohibit the reuse of old versions of keys. This is accomplished by providing the key with a 'disable' field that is activated as soon as a new key with the same key number is addressed.

Table 4.18 Typical key parameters stored in a smart card

Data object	Remarks
Key number	Key reference number; unique within the key file.
Version number	Version number of the key; which may affect key derivation.
Application purpose	Identifies the cryptographic algorithms and the procedures with which the key may be used.
Disable	Allows the key to be temporarily or permanently disabled.
Retry counter	This counter keeps track of non-successful attempts to use the key with a cryptographic procedure.
Maximum retry count	If the retry count reaches the maximum count, the key is blocked.
Key length	—
Key	The actual key.

4.8.6 Key management example

Here we would like to describe an example of key management for a system based on smart cards. The objective is to further illustrate the previously described principles by means of an easily understood general example. Compared with this example, large real systems frequently have arrangements that are much more complex, with several structural layers. Small systems often have no key hierarchy at all, since a secret global key is used for all cards. The system presented here occupies a middle position between systems with very simple structures and large systems, and thus represents a good example.

In the example shown in Figure 4.40, the keys for loading and paying can be used with an electronic purse. They use symmetric cryptographic procedures. These keys are evidently important within the system, since they are relatively well protected by the described key hierarchy. The individual derivation functions are not shown in detail here, but the DES or triple-DES algorithm could always be used for them. The lengths of the keys are also not dealt with in detail, but they certainly may vary. The keys at the top of the hierarchy are normally derived using more powerful cryptographic functions than those used at the lower level keys, for reasons of security.

The key at the top of the hierarchy is called the *general master key*. There is only one such key for an entire generation of keys. A generation could remain valid for a year, for example, and be replaced in the following year by a new generation, which means a new generation of the general master key. The general master key is the most sensitive key of the system with regard to security. If it becomes known, all of the keys of its generation can be computed, and the system is broken for one generation. The general master key may be generated from a random number. It is also conceivable to base the general master key on the values shown by dice thrown by several independent persons, each of whom consequently knows only part of the value of the key. The general master key should never be completely known by any single person, and its generation must under no circumstances be reproducible.

A master key for each function is separately derived from the general master key. These keys may be used for loading or paying with an electronic purse, for example. A one-way function, such as a modified triple-DES algorithm, is used in our example to derive the separate master keys for the various functions. This makes it impossible to compute the general master key from

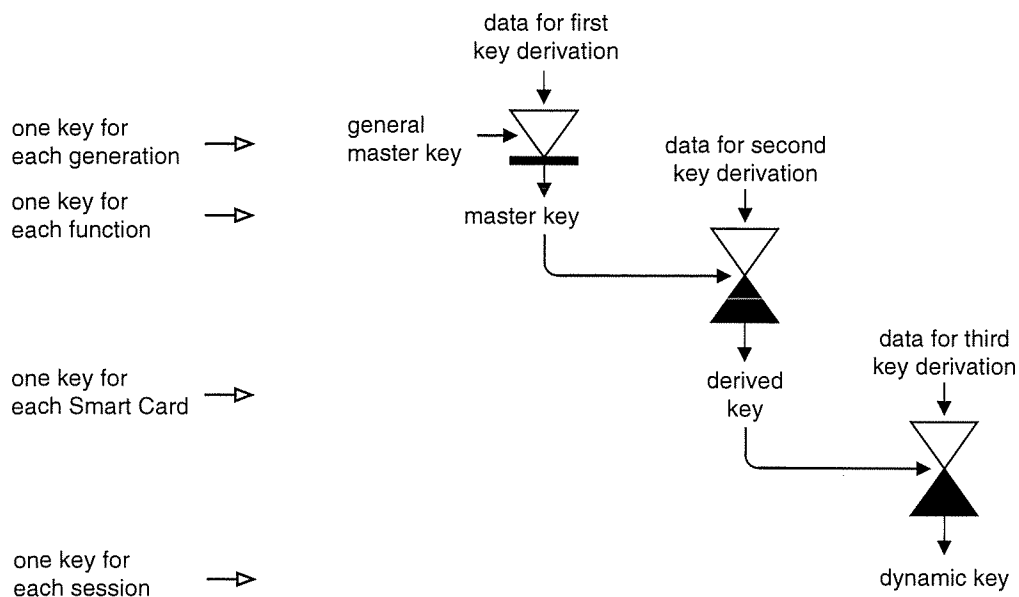


Figure 4.40 An example of a key hierarchy in a system based on smart cards and using symmetric cryptographic algorithms

a master key by applying the procedure in reverse. If a one-way function is not used to derive the master keys, the general master key could be computed if, despite all security measures, a master key becomes known and the derivation parameters are also known. A one-way function is used here because it is assumed that in this imaginary purse system, the master keys will be located in the security modules of local terminals. This means that with regard to system security, they are much more endangered than the general master key, which never leaves the background system.

The derived keys form the next level in the key hierarchy. These are the keys that are located in the smart cards. Each card contains a set of derived keys, which are classified according to their functions and generations. If such a card is used at a terminal, the terminal can compute the derived key for itself, based on the parameters used to derive the key in question. Naturally, the terminal first reads the derivation parameters from the card. Once the derived key is available, the following step is to compute the dynamic key, which is specific to a particular session. This key is valid only for the duration of a single session. The duration of a session ranges from a few hundred milliseconds to a few seconds in most smart card applications. A dynamic key is no longer used after the end of the session.

This example system may appear complicated at first glance, but it is relatively simple compared with real systems. The objective of the example is to show exactly how all the keys in a system can be generated. It also implicitly shows what measures must be taken if a key becomes known. If the general master key becomes known, a switch to a new generation must be made if the system is to continue to be used without concerns about security risks. By contrast, if a derived key becomes known, all that is necessary is to block the card in question; any other key management changes would surely be inappropriate. Of course, all of these measures presume that the reason why one or more keys have become known can be determined, so that it can be prevented in the future.

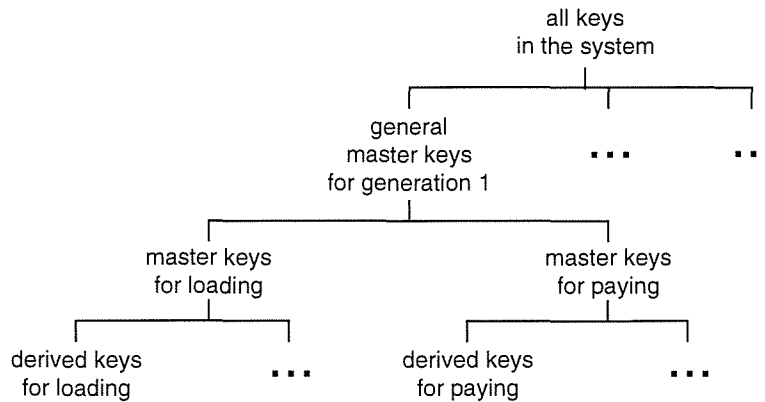


Figure 4.41 Examples of keys for an electronic purse system with two functions: loading and paying. Only the stored keys are shown here; keys that are dynamically generated for individual sessions have been omitted to simplify the diagram

Given this key hierarchy, it is evident that very many keys must be generated and stored in the smart cards. Of course, it is always possible to assign several functions to a single key in order to save memory space. It is also quite conceivable to use a different structure for the key hierarchy, which naturally strongly depends on the system for which the key management system is developed.

4.9 HASH FUNCTIONS

Even powerful computers require a great deal of time to compute a digital signature. In addition, large documents would need many signatures, since the document to be signed cannot be arbitrarily long. A trick is therefore used. The document is first compressed to a much shorter fixed length, and then the signature of the compressed data is computed. It does not matter whether the compression can be reversed, since the signature can always be reproduced from the original document. The functions used for this type of computation are called one-way hash functions.

Generally speaking, a one-way hash function is a function that derives a fixed-length value from a variable-length document in a manner such that this value represents the original content of the document in a compressed form and cannot be used to reconstruct the original document. In the smart card domain, these functions are used exclusively to compute the input values for digital signatures. If the length of the document is not a multiple of the block length used by the hash function, it must be padded appropriately.

For a hash function to be effective, it must exhibit certain properties. The result must have a fixed length, so that it can be readily used by signature algorithms. Since large quantities of data normally have to be processed, the hash function must have a high throughput. It must also be easy to compute the hash value. By contrast, it should be difficult, or better yet impossible, to derive the original document from a known hash value. Finally, the hash function must be collision-resistant. This means that for a given document, it should not be easy to find a second document that yields the same hash value. Nevertheless, there certainly will be other

documents with the same hash value. This is only natural, since all possible messages, ranging in length from null to infinity, are represented by a set of hash values having the same fixed length. An unavoidable consequence of this is that collisions will occur. That is why the term 'collision-resistant' is used, rather than 'collision-free'.

What is the effect of a collision? There will be two different documents with the same hash value, and thus the same digital signature. This will have the fatal consequence of making the signature worthless, since it would be possible to alter the document without anyone being able to detect the fact. This is precisely what is involved in one of the two typical attacks on hash functions, which consists of systematically searching for a second document that has the same hash value as the original document. If the content of this document makes sense, the digital signature derived from the hash value is discredited. Since the two documents are interchangeable, the signature is worthless. After all, it makes an enormous difference whether a house purchase contract is for €10,000 or €750,000.

The second type of attack on a hash value is somewhat subtler. In this case, two documents with identical hash values but different contents are prepared in advance. This is not particularly difficult, considering all the special symbols and extensions available in the character set. The result is that a single digital signature is valid for both documents, and it is impossible to prove which document was originally signed.

Finding two documents with the same hash value is not as difficult as it might seem. It is possible to exploit the birthday paradox, which is well known in statistical theory. This paradox involves two questions. The first question is: how many people must be in a room for the probability to be greater than 50 % that one of them has the same birthday as the person asking the question. The answer can be easily found, since it is only necessary to compare the birthday of the questioner with the birthday of everyone else in the room. There must be at least 183 ($365 \div 2$) people in the room.

The second question reveals the paradox, or better, the surprising result of this comparison. This question is: how many people must be in a room for the probability to be greater than 50 % that two people in the room have the same birthday. The answer is only 23 people. The reason is that although only 23 people are present, this represents a total of 253 pairs for comparing birthdays. The probability that two people have the same birthday is based on these pairs.

Precisely this paradox is utilized in attacking a hash function. It is much easier to create two documents that have the same hash value than it is to modify a document until it yields a given hash value. The consequence is that the results of hash functions must be large enough to successfully foil both types of attack. Most hash functions thus produce values that are at least 128 bits long, which is presently generally considered to be adequate with regard to the two types of attack just described.

Many different hash functions have been published up to now, and some of them are also defined in standards. However, these functions are frequently modified as a consequence of the discovery of a successful form of attack. Table 4.19 provides a short summary of the hash functions currently in common use. Unfortunately, a description of their internal operation is beyond the scope of this book.

The ISO/IEC 10118-2 standard specifies a hash function based on an n -bit block-encryption algorithm (e.g. DES). With this algorithm, the length of the hash value may be n or $2n$ bits. The MD4 (message digest 4) hash function (presently rarely used) and its successor MD5 were published by Ronald L. Rivest in 1990–1991. They are based on a standalone algorithm, and both functions generate a 128-bit hash value. In 1992, the NIST published a hash function

Table 4.19 Summary of commonly used hash functions

Name	Input block size	Hash value length
ISO/IEC 10118-2	n bits (e.g. 64 or 128 bits)	n or $2n$ bits (e.g., 64 or 128 bits)
MD4	512 bits	128 bits
MD5	512 bits	128 bits
MDC-4	64 bits	128 bits
RIPEMD-128	512 bits	128 bits
RIPEMD-160	512 bits	160 bits
SHA-1	512 bits	160 bits

for the DSS algorithm that is known as SHA. After the discovery of certain weaknesses, it was modified, and the resulting function has been known since mid-1995 as SHA-1. It is also standardized under the name FIPS 180–1.

Since data transmission to smart cards is generally slow, the hash function is performed in the terminal or in a computer connected to the terminal. This drawback is balanced by the fact that this makes the hash function interchangeable. Besides, in most cases, memory limitations prevent hash functions from being stored in the cards. The program size is in almost every case around 4 kB of assembler code. The throughput of typical hash functions is very high relative to the demands placed on them. With an 80386 computer running at 33 MHz, it is usually at least 300 kB/s, and it lies in the range of 4 to 8 MB/s with a 200-MHz Pentium PC.

4.10 RANDOM NUMBERS

Random numbers are repeatedly needed in connection with cryptographic procedures. In the field of smart cards, they are typically used to ensure the uniqueness of a session during authentication, as padding for data encryption and as initial values for send sequence counters. The length of the random number needed for these functions usually lies in the range of 2 to 8 bytes. The maximum length naturally comes from the block size of the DES algorithm.

The security of all these procedures is based on random numbers that cannot be predicted or externally influenced. The ideal solution would be a hardware-based random number generator in the card's microcontroller. However, this would have to be completely independent of external influences, such as temperature, supply voltage, radiation and so on, since otherwise it could be manipulated. That would make it possible to compromise certain procedures whose security relies on the randomness of the random numbers. Current random number generators in smart card microcontrollers are generally based on linear feedback shift registers (LFSRs) driven by voltage-controlled oscillators.

Even with the current level of technological development, it is difficult to construct a random number generator immune to external influences (a 'true random-number generator', or TRNG) in silicon on a microcontroller die. Consequently, operating system designers frequently take recourse to software implementations. These yield pseudo-random number generators (PRNGs), most of which produce very good (that is, random) random numbers. Nevertheless, they do not generate truly random numbers, since the numbers are computed

using strictly deterministic algorithms and thus can be predicted if the algorithm and its input values are known. This is why they are called ‘pseudo-random numbers’.

It is also very important to ensure that the cards of a production batch generate different sequences of random numbers, so that the random numbers produced by one card cannot be inferred from those produced by another card from the same batch. This is achieved by entering a random number as the seed number (starting value) for the random number generator when the operating system is completed in each card.

4.10.1 Generating random numbers

There are many different ways to generate random numbers using software. However, since the memory capacity of smart cards is very limited and the time needed to perform the computation should be as short as possible, the number of options is severely restricted. In practice, essentially only methods that utilize functions already present in the operating system are used, since they require very little additional program code.

Naturally, the quality of the random numbers must not be adversely affected if a session is interrupted by a reset or by removing the card from the terminal. In addition, the generator must be constructed such that the sequence of random numbers is not the same for every session. This may sound trivial, but it requires at least a write access to the EEPROM to store a new seed number for the next session. The RAM is not suitable for this purpose, since it needs power to retain its contents. One possible means of attack would be to repeatedly generate random numbers until the EEPROM cells holding the seed number fail. Theoretically, this would cause the same sequence of random numbers to then occur in every session, which would make them predictable and thus give the attacker an advantage. This type of attack can easily be averted by constructing the relevant part of the EEPROM as a ring buffer and blocking all further actions once a write error occurs.

Another very important consideration for a software random number generator is to ensure that it never runs in an endless loop. This would result in a markedly shorter repeat cycle for the random numbers. It would then be easy to predict the numbers, and the system would be broken.

Almost every smart card operating system includes an encryption algorithm for authentication. It is an obvious idea to use this as the basis for a random number generator. In this regard, it is important to realize that a good encryption algorithm mixes the plaintext as thoroughly as possible, so that the plaintext cannot be derived from the ciphertext without knowledge of the key. A principle known as the avalanche criterion says that, on average, changing one input bit should change half of the output bits. This property can be usefully exploited for generating random numbers. The exact structure of the generator depends on the specific implementation.

Figure 4.42 illustrates a possible arrangement. This generator uses the DES algorithm with a block length of 8 bytes, with the output value being fed back to the input. Naturally, any other encryption algorithm could also be used. The generator works essentially as follows. The value of a ring buffer element is encrypted by DES using a key unique to the card. The ciphertext so produced is the 8-byte random number. This number, when XORed with the previous plaintext, provides the new entry for the EEPROM ring buffer. The generator then moves to the following entry in the cyclic ring buffer. This relationship can be expressed mathematically as $RND_n := f(key, RND_{n-1})$.

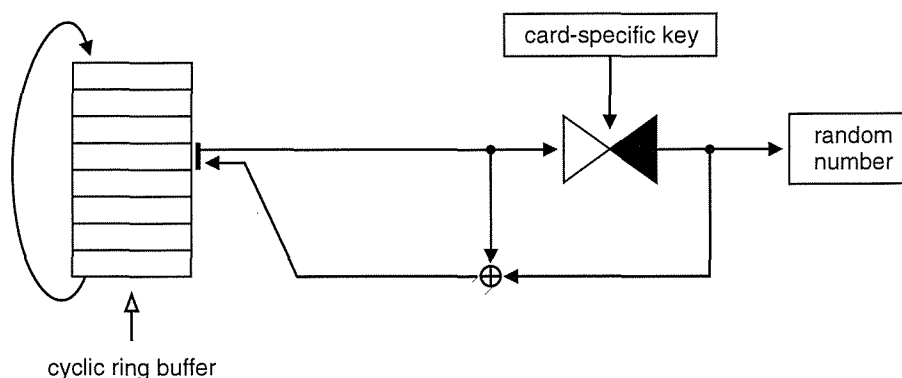


Figure 4.42 Sample architecture of a DES pseudo-random number generator for smart card operating systems. This generator is primarily designed to minimize the number of write accesses to the EEPROM

When the smart cards are completed, a card-specific DES key is stored in each card, and at the same time random seed numbers are entered into the ring buffer, which for example could be a 12×8 buffer. The seed numbers ensure that each card produces a unique sequence of random numbers. A 12-stage ring buffer increases the life span of the generator by a factor of 12. Assuming that the EEPROM is guaranteed to have 100,000 write cycles, this generator can produce at least 1,200,000 8-byte random numbers.

Erasing and writing eight bytes in the EEPROM takes about 14 ms ($2 \times 2 \times 3.5$ ms), and executing the DES algorithm takes about 17 ms at 3.5 MHz if it is implemented in software. The remaining processing time is negligible. The card thus needs around 31 ms to generate a random number. However, if the DES algorithm is computed in hardware (at a typical rate of 0.1 ms/block), a random number could be generated in only 14.4 ms using the described method.

Figure 4.43 shows another example of a pseudo-random number generator. This generator is initialized every time the card is reset, which is the only time a write access to the EEPROM occurs. Only RAM accesses are used for the subsequent generation of random numbers, which makes this generator relatively fast. However, the disadvantage of this is that the generator uses a few bytes of RAM for the duration of the session. The statistical quality of this pseudo-random number generator is not very good, but it is adequate for normal smart card authentication procedures. The primary consideration with such procedures is to avoid generating random numbers with short repeat cycles, since that would allow authentication to be compromised by replaying messages from previous sessions.

The FIPS 140-2 standard recommends that security modules check their built-in random number generators after every reset using statistical tests. Only after these tests have been successfully completed should the random number generator be released for further use. Current commonly used smart card operating systems rarely include such capability, since it is assumed that due to the deterministic nature of the pseudo-random number generator, the statistics of the generated random numbers will not change significantly.

The number of proposals, standards and designs for pseudo-random number generators is simply overwhelming. Some well-known examples are the generators in the X9.17 standard, FIPS 186, the proposals in the Internet RFC 1750 and the arrangements shown by Bruce Schneier [Schneier 96], Peter Gutmann [Gutmann 98a] and Benjamin Jun [Jun 99]. The guiding

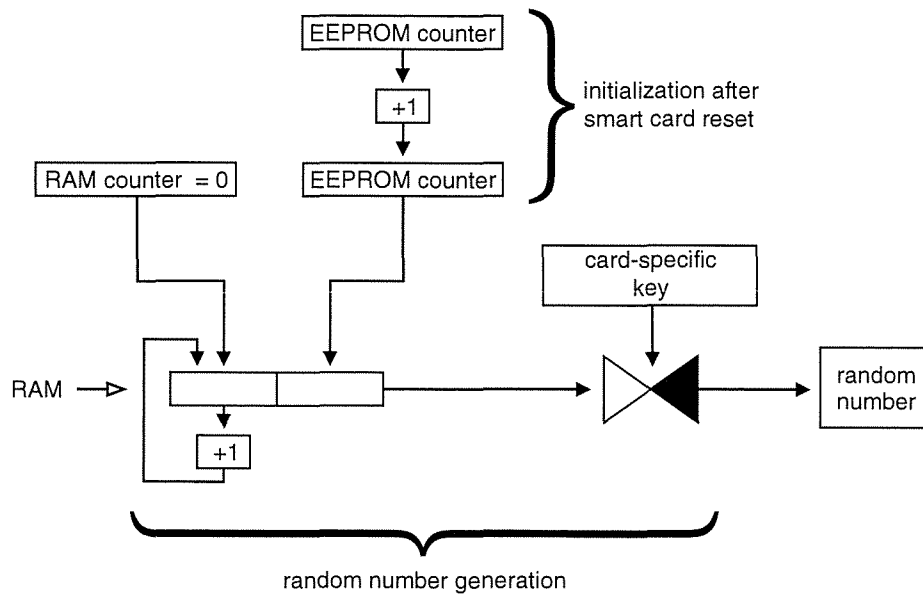


Figure 4.43 Sample architecture of a DES pseudo-random number generator for smart card operating systems. This generator is faster than the one shown in Figure 4.42, since only one EEPROM write cycle is needed per session. The quality of the random numbers it produces is adequate for normal smart card applications (authentication using the challenge–response procedure)

principle for a random number generator should always be to keep it as simple and easily understandable as possible. Only then is it possible to assess its characteristics and thus determine its quality.

4.10.2 Testing random numbers

After a random number generator has been implemented, it is generally necessary to test the quality of the numbers it produces. Fundamentally, there should be a nearly equal number of ones and zeros in the generated random numbers. However, it is not enough to simply print out a few numbers and compare them. Random numbers can be mathematically tested using standard statistical procedures. It is self-evident that a large number of 8-bit random numbers will be needed for such testing. Between 10,000 and 100,000 random numbers should be generated and analyzed in order to arrive at reasonably reliable results. The only way to test this many numbers is to use computerized testing programs.

When evaluating the quality of the random numbers, it is also necessary to investigate the distribution of the generated numbers. If this is very uneven, with certain values strongly favored, then exactly these regions can be used for purposes of prediction. This means that Bernoulli's theorem should be satisfied as closely as possible. This theorem states that the occurrence of a particular number, independent of what has come before it, depends only on the probability of occurrence of the number itself. For example, the probability that a 4 appears when a die is thrown is always $1/6$, independent of whatever number appeared on the previous throw. This is also referred to as 'event independence'.

The period of the random numbers, which is the number of random numbers generated before the series repeats itself, is also very important. It must naturally be as long as possible, and in any case longer than the lifetime of the random number generator. In this way, the possibility of attacking the system by recording all random numbers generated for a complete period can be excluded in a quite simple and reliable manner.

There are many statistical tests for investigating the randomness of events, but in practice, we can limit ourselves to a few simple tests whose results are easily interpreted. There are also many publications on the subject of testing for randomness [Knuth 97, Menezes 97], as well as corresponding standards [FIPS 141-2, RFC 1750]. One test that is simple to set up and easy to interpret is to count the number of times that each byte value occurs in a large number of random numbers. If the results are displayed graphically as shown in Figure 4.44, they give a good indication of the distribution of the numbers.

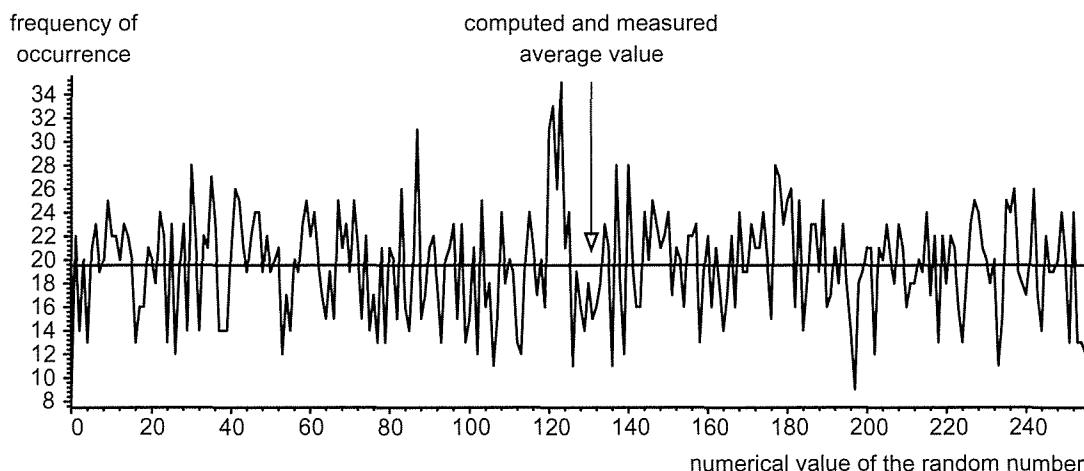


Figure 4.44 Statistical distribution of a series of 5000 single-byte random numbers. This is also referred to as the spectral distribution over one byte. These numbers were generated by a typical smart card pseudo-random number generator. Based on purely mathematical considerations, each of the possible values (in the range of 0–255) should occur 19.5 times

If such a diagram is used to investigate 8-byte random numbers, the values plotted on the horizontal axis must still be single-byte or at most two-byte numbers, since the number of samples needed for a statistical analysis would otherwise become extremely large. A good guideline is that every random number should occur approximately four to 10 times for each value in order to obtain reasonably reliable results. In this way, it is possible to quickly see whether the random numbers that have been generated fully exploit the possible bandwidth of the byte. If certain values are strongly favored, this offers an attacker a possible starting point.

Unfortunately, this test does not say anything about the order in which the random numbers occur, but only something about their distribution. For example, it would be possible for a ‘random number’ generator to output numbers cyclically from 0 to 255. This would yield an outstandingly uniform distribution, but the numbers would be completely predictable. Other tests must be used to assess this quality criterion for random numbers.

Another practical test that yields a simple and quick estimate of the quality of a series of random numbers is to compress the series using a file-compression program. According to

Shannon, the degree of compression that is possible is inversely related to the randomness of the set of generated numbers.

A significantly more robust test is the very well-known χ^2 test. Although it tests the same aspect as the previously described graphic test for even-statistical distribution, it is significantly more exact because it is performed using a mathematical procedure [Bronstein 96]. If the random numbers are assumed to be evenly distributed, the median value and standard deviation can be calculated. The deviation from a normal distribution can then be determined based on a χ^2 distribution. From this, it is possible to state a numerical value for the distribution of the random numbers.

However, this test cannot be used to draw any conclusions regarding the sequence in which the random numbers occur. Other statistical tests can be used to verify the randomness with which the numbers occur [Knuth 97], such as the Serial Test, which analyzes the periods of patterns that occur in the random numbers. Similarly, the Gap Test analyzes the intervals over which patterns do not occur. The Poker Test should also be used to evaluate the χ^2 distribution of patterns that do occur, and the Coupon Collector Test should be used to evaluate the χ^2 distribution of patterns that do not occur.

The Spectral Test, which investigates the relationship between each random number and the next following number, also has a certain amount of relevance [Knuth 97]. In the two-dimensional version of this test, random numbers and their immediate successors are plotted in an X-Y coordinate system, as shown in Figure 4.45. The three-dimensional version requires the successor to the successor number in addition, as well as a third axis (the Z axis). *N*-dimensional spectral tests can be performed in a similar manner, but for understandable reasons, they must dispense with graphical representation.

At a minimum, the above-mentioned tests must be performed and analyzed in order to achieve a reliable and definitive evaluation of a random number generator. Additional calculations and tests can be used to confirm the results so obtained. Only in this way is it possible to make a reasonably correct assessment of the quality of a set of random numbers.

subsequent
random number

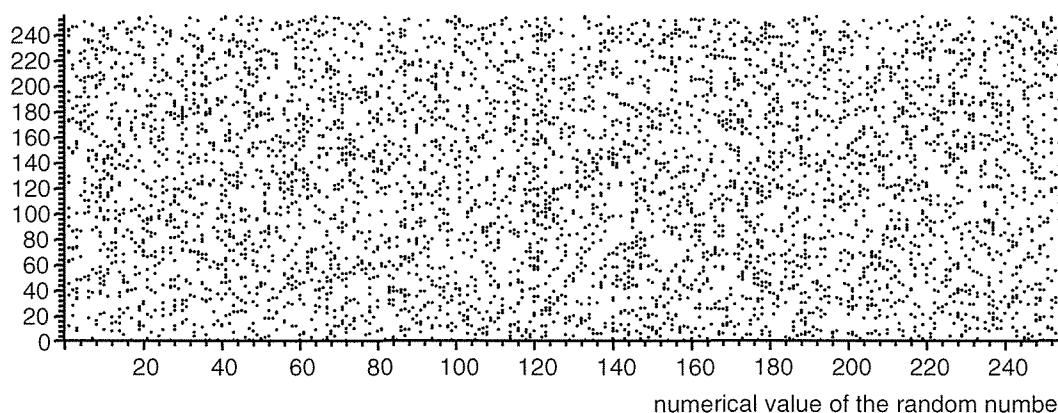


Figure 4.45 Graphic representation of the distribution of successor values of 5000 single-byte random numbers, corresponding to a spectral test. The nearly uniform distribution of the successor values can be seen at a glance from the regular pattern. The numbers were generated by a typical smart card pseudo-random number generator

Of course, considering the areas in which random numbers are used in smart card applications, an overly sophisticated random number generator is usually not justified. For instance, the effect on security of being able to predict the random numbers used for authentication would be very slight, since no attack is possible without knowledge of the private key used to encrypt the random number.

A more serious problem would, however, arise if it were possible to manipulate the random number generator, for example so that it would always generate the same sequence of random numbers. In this case, an attack based on replaying the numbers would be not only possible but also successful. This would also be true if the period of the random numbers were very short. In each individual case, the primary conditions that the random numbers must satisfy must be carefully considered, since this naturally affects the random number generator. Although a supreme effort here may lead to very high-quality random numbers, it also usually results in increased use of memory space, which is particularly limited in smart cards.

Table 4.20 Summary of standard statistical tests for random numbers

Test and reference	Remarks
Coupon collector test [Knuth 97]	χ^2 distribution of the non-occurrence of
Poker test [Menezes 97]	patterns in a series of random numbers.
Frequency test [Knuth 97, Menezes 97]	Counting the number of ones in a series of random numbers.
Gap test [Knuth 97]	Investigating the patterns that do not occur in a series of random numbers.
Long run test per FIPS 140-2	Investigating whether a series of ones and zeros with a length of 34 bits occurs in a series of random numbers that is 20,000 bits long.
Monobit test per FIPS 140-2	Counting the number of ones in a series of random numbers that is 20,000 bits long.
Poker test [Knuth 97]	χ^2 distribution of the occurrence of patterns in a series of random numbers.
Poker test per FIPS 140-1	Counting 4-bit patterns in a series of random numbers that is 20,000 bits long.
Serial test [Menezes 97]	Investigating maximum length of a series of all ones or all zeros in a series of random numbers that is 20,000 bits long.
Runs test per FIPS 140-1	Investigating the patterns that occur in a series of random numbers.
Serial test [Knuth 97]	Investigating the distribution of successor values of random numbers.
Spectral test [Knuth 97]	

4.11 AUTHENTICATION

The purpose of authentication is to verify the identity and genuineness of a communications partner. Translated into the world of smart cards, this means that the card or the terminal determines whether its communications partner is a genuine terminal or a genuine smart card,

respectively. For the sake of clarity, the term ‘identification’ is consistently used in this book to refer to verifying the authenticity of persons, although in principle it falls under the general concept of authentication.

Authentication requires the communicating parties to share a common secret that can be verified by means of an authentication procedure. Such a procedure is significantly more secure than a pure identification procedure, such as a PIN test. In the latter case, all that happens is that a secret (the PIN) is sent to the card, which confirms its genuineness if it is correct. The drawback of this procedure is that the secret is sent as plaintext to the card, which means that an attacker could easily come to know the secret (the PIN).

By contrast, with an authentication procedure it is not possible to discover the common secret by tapping the communications channel, since the secret does not have to be sent openly via the interface. A distinction is also made between static and dynamic authentication. In a static procedure, the same (static) data are always used for the authentication. A dynamic procedure, by contrast, is constructed such that it is protected against being attacked by re-entering data recorded during a previous session. This is because each authentication is based on different data when dynamic authentication is used.

There is also a fundamental difference between unilateral and mutual authentication procedures. A unilateral authentication, if it is successful, establishes the authenticity of one of the two communications partners. Mutual authentication, when successful, establishes the authenticity of both of the communications partners.

Authentication procedures based on cryptographic algorithms and used with smart cards can be further classified into symmetric and asymmetric procedures. Currently, the procedures used with smart cards are almost exclusively symmetric. Due to their slow execution speeds, asymmetric procedures, which means those based on the RSA algorithm or similar algorithms, do not yet have any practical significance with regard to smart cards systems. However, it can be foreseen that this will change in the future. In any case, the operating principle of asymmetric procedures is the same as that of symmetric procedures.

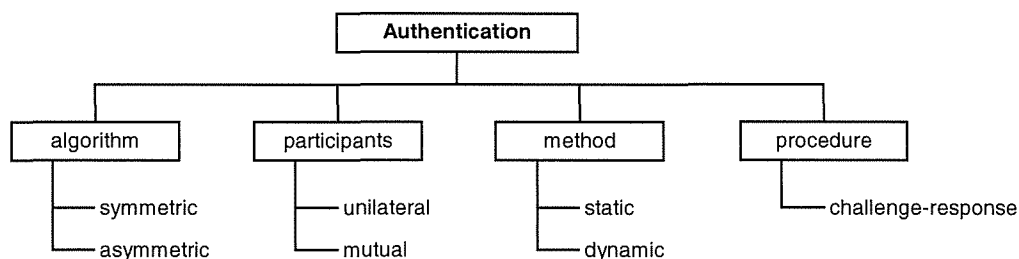


Figure 4.46 Classification of authentication procedures used with smart card systems

There are several standards relating to the authentication of equipment. The ISO/IEC 9798 standard is the most prominent of these. Part 2 of this standard describes symmetric procedures, while Part 3 describes asymmetric procedures. Fundamentally, the five parts of the ISO/IEC 9798 standard form an outstanding compilation of the commonly used authentication procedures, including symmetric, asymmetric, MAC-based and zero-knowledge-based procedures.

The principle of authentication in the field of smart cards is always based on a challenge-response procedure. In this procedure, one of the communications partners first asks the other one a randomly generated question (the challenge). The second partner computes an answer

using an algorithm and sends the answer (the response) back to the first one. Naturally, the algorithm is preferably an encryption using a shared secret key that represents the common secret of the two communications partners.

4.11.1 Symmetric unilateral authentication

A unilateral authentication serves to assure one party of the trustworthiness of the other party to a communication. For it to be possible, both parties must have a shared secret, the knowledge of which is verified by the authentication procedure. This secret is the key for an encryption algorithm, and the entire security of the authentication procedure depends on this key. If the key should become known, an attacker could authenticate himself just as readily as a genuine communications partner.

The principle of unilateral authentication with a symmetric cryptographic algorithm is illustrated in Figure 4.47. For the sake of clarity, it is assumed that the terminal authenticates a smart card. This means that the terminal determines whether the smart card is trustworthy.

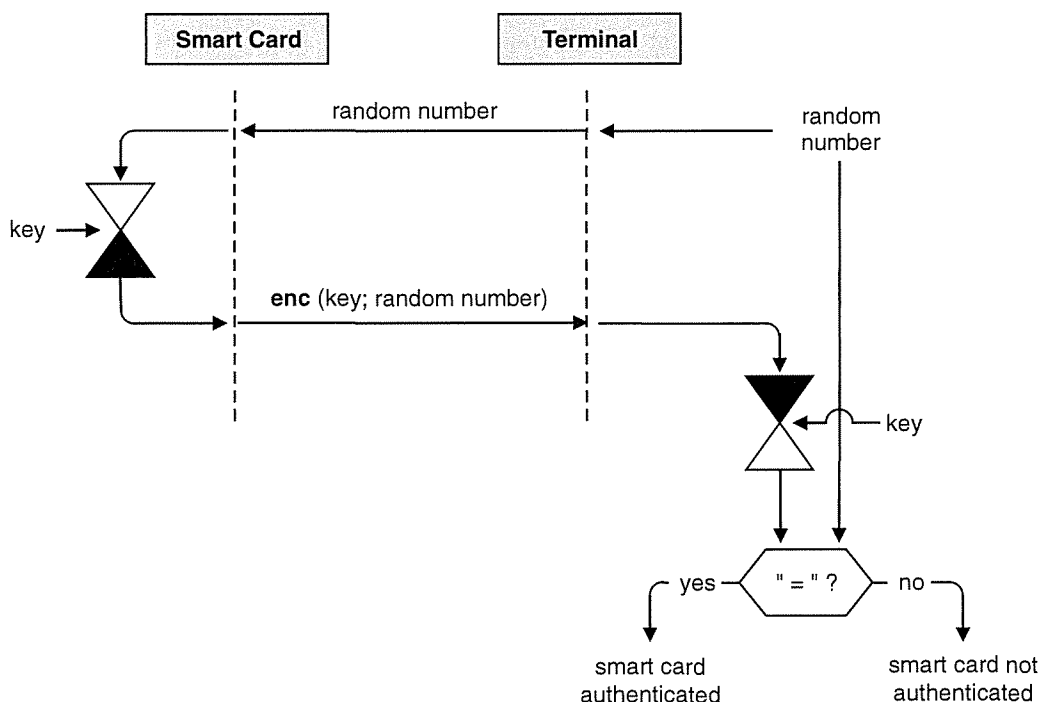


Figure 4.47 Working principle of unilateral authentication with a symmetric cryptographic algorithm. This example shows the authentication of a smart card by a terminal, which can be implemented using the INTERNAL AUTHENTICATE command of the ISO/IEC 7816-4 standard

The terminal generates a random number and sends it to the smart card. This is the challenge. The smart card encrypts the random number it receives, using a key known to both the card and the terminal. The security of the procedure depends on this key, since only the possessor of the secret key can generate the correct response to be sent to the terminal. The card then returns the result of the encryption to the terminal. This is the response to the challenge. The

terminal uses the secret key to decrypt the encrypted random number it has received, and then compares the result with the random number it originally sent. If the two numbers match, the terminal knows that the smart card is authentic.

This procedure cannot be attacked by replaying a challenge or response that has been intercepted from an earlier session, since a different random number is generated for each session. The only type of attack with a moderately good chance of success would be to systematically search for the secret key. Since the challenge and response are simply a plaintext-ciphertext pair, the secret key could be discovered using a brute-force attack.

If all the cards for a given application have the same key and this key becomes known, the entire system will be discredited. In order to avoid exactly this possibility, in practice only card-specific keys are used as a matter of principle. This means that every card has an individual key, which may be derived from a non-secret feature of the card. This specific feature can be the serial number of the chip, which is written to the chip when it is manufactured, or some other number that is specific to each card.

In this case, the terminal requests the chip number from the smart card in order to compute the card-specific key. The chip number is specific to the card and unique within the system, so there is no other card in the system that matches this card. The value of the card-specific secret key is a function of the card number and the master key, which is known to the terminal. In practice, a portion of the card number is encrypted using the master key, and the result is used as the card-specific authentication key. A DES or triple-DES algorithm can be used for the encryption.

It must of course be borne in mind that if the master key (which is known only to the terminal) becomes compromised, the entire system will be compromised, since all card-specific authentication keys can be computed using the master key. The master key must therefore be securely stored in the terminal (in a security module, for example), and, if possible, it should be actively erasable in case of an attack.

Once the terminal has computed the necessary authentication key for the card, the usual challenge-response procedure occurs. The smart card receives a random number, encrypts it using its individual key and returns the result to the terminal. The terminal executes the reverse function of the computation performed by the card and compares the two results. If they match, the terminal and the smart card have a common secret, which is the secret card-specific key, and the smart card has been authenticated by the terminal.

In this case, the authentication process is somewhat time-intensive due the use of the DES algorithm (to the extent that it is implemented in software) and the data transmission from and to the card. This can cause problems in some applications. Given certain assumptions, we can roughly calculate the time required to perform a unilateral authentication. We assume that the smart card has a 3.5-MHz clock, uses the $T = 1$ transmission protocol, has a divisor of 372 and uses a DES algorithm that takes 17 ms per block. Without going into details, we assume that the internal routines in the smart card take 9 ms. This simplifies the calculation without significantly distorting the result, which is shown in Table 4.21. As can clearly be seen from this calculation, a single authentication takes around 65 ms. This will not usually cause any time-related problems in an application.

4.11.2 Symmetric mutual authentication

The principle of mutual authentication is based on dual unilateral authentication. In principle, two successive unilateral authentications could also be used, one for each of the communicating

Table 4.21 Calculation of the processing time within a smart card for a unilateral authentication, taking data transmission times into account

Command	Data transmission		Computation	Total
INTERNAL AUTHENTICATE (DES in software)	38.75 ms	+	26 ms	= 64.75 ms
INTERNAL AUTHENTICATE (DES in hardware)	38.75 ms	+	0.08 ms	= 38.83 ms

parties, in order to achieve mutual authentication. However, since the communications overhead must be kept as low as possible to minimize the time required by the process, a procedure that interleaves the two unilateral authentication processes has been defined. This also increases the security of the procedure, since it is much more difficult for an attacker to intervene in the communications process.

Before the terminal can compute the card-specific authentication key from the card number, it first needs the card number. After the terminal has received this number, it computes the specific authentication key for this card. It then requests a random number from the card, and at the same time it generates a random number itself. The terminal then swaps the two random numbers and concatenates them, after which it encrypts the resulting number using the authentication key. Finally, it sends the resulting ciphertext to the card. The objective of reversing the random numbers is to allow the challenge and response to be distinguished from each other.

The card can decrypt the received block and check whether the random number it previously sent to the terminal matches the number it received in return. If this is the case, the smart card knows that the terminal possesses the secret key. This authenticates the terminal with respect to the card. Next, the smart card swaps the two random numbers, encrypts the resulting number using the secret key and sends the resulting ciphertext block back to the terminal.

The terminal decrypts the received block and compares the random number it previously sent to the card with the one it has received in return. If they match, the smart card has been authenticated with respect to the terminal. This completes the mutual authentication process, and the terminal and the smart card both know that the other is trustworthy.

To minimize the communications time, the smart card can return the random number together with its card number. This is particularly attractive when mutual authentication takes place between a smart card and a background system. In this case, the card is directly addressed by the background system, with the terminal being 'transparent'. The data transmission rate in such situations is often very low, so the communications process must be streamlined as much as possible.

In order to illustrate the considerable amount of time required for a mutual authentication compared with a unilateral authentication, we can again make a sample calculation. The basic assumptions are the same as for the calculation of the time required for unilateral authentication (see Table 4.21). The results are shown in Tables 4.22 (for software implementations) and 4.23 (for hardware implementations). As can be seen, mutual authentication takes nearly three times as long as unilateral authentication.

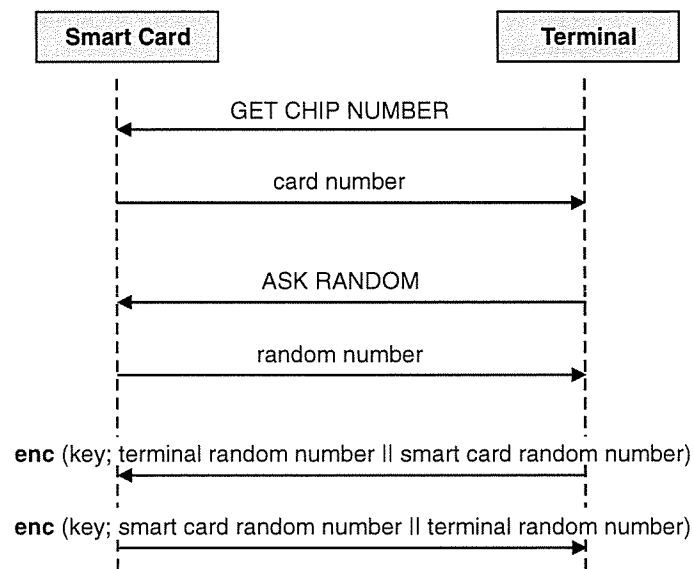


Figure 4.48 Mutual authentication using a card-specific key and a symmetric cryptographic algorithm. The illustrated procedure corresponds to a mutual authentication of a smart card and a terminal as implemented in the ISO/IEC 7816-8 AUTHENTICATE command

Table 4.22 Estimated time required for a smart card to perform mutual authentication if random number generation and DES computation are implemented in software, including data transmission time. It is assumed that derived keys are not used, so GET CHIP NUMBER is not necessary

Command	Data transmission	Computation	Total
ASK RANDOM	28.75 ms	26 ms	
MUTUAL AUTHENTICATE	68.75 ms	95 ms	
	97.50 ms	+	121 ms = 218.50 ms

Table 4.23 Estimated time required for a smart card to perform mutual authentication if random number generation and DES computation are implemented in hardware, including data transmission time. It is assumed that derived keys are not used, so GET CHIP NUMBER is not necessary

Command	Data transmission	Computation	Total
ASK RANDOM	28.75 ms	0.08 ms	
MUTUAL AUTHENTICATE	68.75 ms	0.16 ms	
	97.50 ms	+	0.24 ms = 97.74 ms

4.11.3 Static asymmetric authentication

Only a few smart card microcontrollers have arithmetic processing units that can be used to execute the RSA algorithm. This is mainly because such capability would take up additional space on the chip, which would increase its price.

However, the fact that a supplementary asymmetric authentication procedure would offer increased protection, since it requires an attacker to break two cryptographic algorithms instead of only one, often makes its use attractive. The problem presented by the absence of a suitable arithmetic processing unit on the card can be dealt with by the expedient of using static authentication of the card by the terminal. This only requires verification within the terminal, and an additional security module in the terminal does not significantly increase its overall cost. This solution is thus much more economical than the use of special smart card microcontrollers. In addition, this procedure is much faster, since only one asymmetric encryption is required, as opposed to two in the case of dynamic asymmetric authentication.

The price of this compromise is reduced security of the authentication procedure. With a static procedure, there is naturally no protection against replaying previous data. This is why it is used only as a supplementary verification of the authenticity of the card, which has already been verified using a dynamic symmetric procedure.

The procedure works essentially as follows. When each smart card is personalized, card-specific information is entered into the card. This can for example be a card number, as well as the name and address of the cardholder. This information does not change during the lifetime of the card. As part of the personalization of the card, the digital signature of this information is computed using a secret key. This key is used globally in the system. When the card is used at a terminal, the terminal reads the signature and the signed data from a file in the card. The terminal has the public key, which is valid for all cards in the system, and it can use this key to encrypt the signature it has read and then compare the result with the data it has read from the card. If these two values match, the card has been authenticated by the terminal.

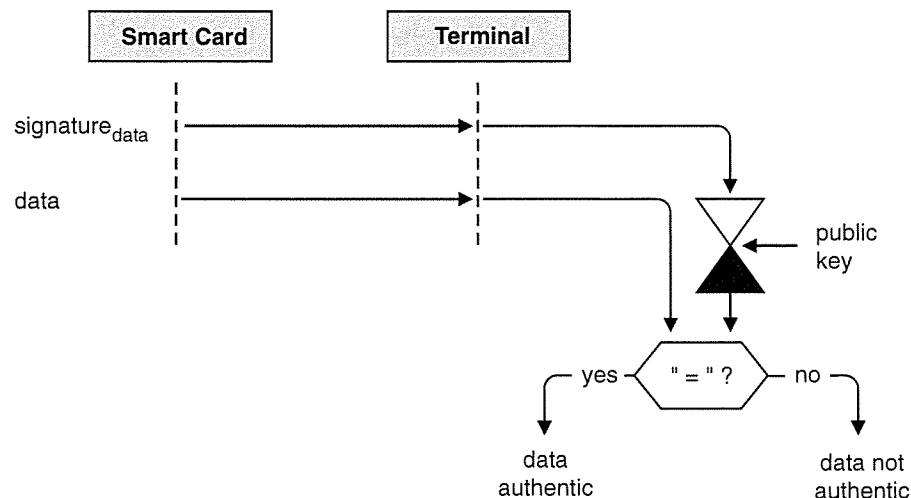


Figure 4.49 Operating principle of static, asymmetric unilateral authentication of a smart card by a terminal using a global key

The procedure illustrated in Figure 4.49, in addition to lacking protection against replaying data, has yet another drawback, which is that a global key is used to generate and verify the signature. Although the key in the terminal does not need to be protected, since it is public, global keys (which are the same for all cards) should fundamentally not be used in a large system. If such a key is broken, or if it becomes known for any other reason, authentication is rendered worthless in the entire system. This means that it is necessary to introduce card-specific key pairs for static authentication.

However, this presents a problem with the memory capacity of the terminals, since each terminal must hold all available public keys for signature verification. Even in a medium-sized system, such as one with one million smart cards, this would require each terminal to have 128 MB of memory for key storage, assuming 1024-bit RSA keys. This would increase the price of the terminals to a level that would not be acceptable to system operators.

When symmetric methods are used, it is quite easy to derive the card-specific keys from a master key.¹⁴ This is not possible with asymmetric methods, due to the way the keys are generated. Consequently, a different approach is taken when card-specific keys are required. The public key for the verification of the signature is stored in the card, along with the signature. In the system of the previous example, the amount of memory needed to store the public keys is still 128 MB, but this is now distributed in 128-byte packets over one million cards. The terminal thus reads the public key from a file in the smart card and can then use it to verify the signature. This avoids the problem of having to store all the public keys of the system in every terminal.

However, an attacker could now generate a key pair and use these keys to sign the information in a counterfeit card. The terminal would read the public key and conclude that the card was genuine. A refinement of the procedure just described is therefore required. This consists of signing the combination of the public key and the card-specific key stored in each card, using a global secret key. This signature is then stored in each card.

The terminal now works as follows. It first reads the public and card-specific keys from the card and then tests the authenticity of the card-specific key using the global public key. If the card-specific key is authentic, the terminal then reads the actual data and verifies them using the public key stored in the smart card. This procedure is shown in Figure 4.50.

These two procedures are already used in some systems, and they will certainly be used increasingly in the coming years. However, as soon as the inclusion of an arithmetic processing unit for asymmetric cryptographic algorithms does not significantly increase the price of a smart card microcontroller, these two procedures will lose a lot of their significance. Their biggest disadvantage is the absence of protection against replaying data from earlier sessions. Although this can be partially compensated by the use of various tricks, such as reusing signed data in subsequent symmetric cryptographic algorithms, it is still not possible to match the level of protection provided by dynamic authentication procedures.

4.11.4 Dynamic asymmetric authentication

All of the previously described static asymmetric procedures have certain disadvantages. These can be eliminated by making the authentication dynamic, which provides protection against

¹⁴ See also Section 4.8.1, 'Derived keys'

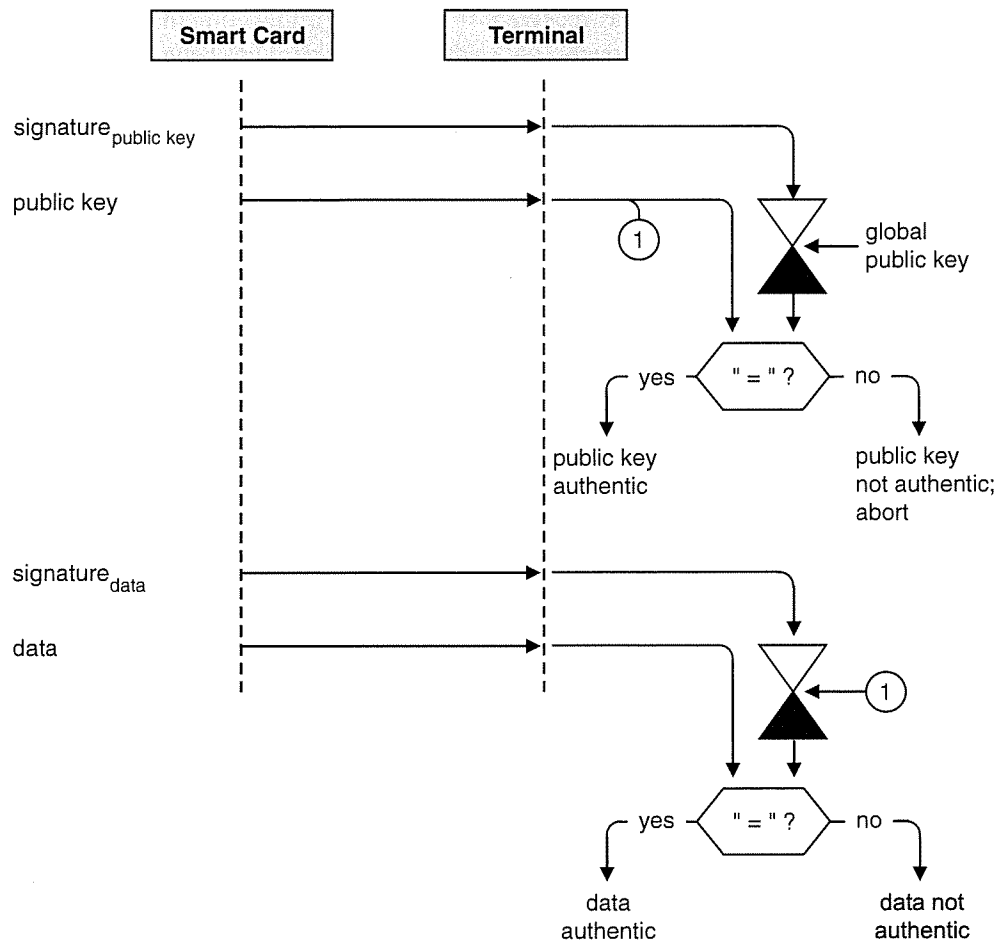


Figure 4.50 Operating principle of static, asymmetric unilateral authentication of a smart card by a terminal using a card-specific key

the re-entry of data intercepted from earlier sessions. The usual practice is to use a random number as the input value for a cryptographic algorithm. Of course, this means that the card must contain an arithmetic processing unit that can execute the asymmetric cryptographic algorithm.

Figure 4.51 illustrates a unilateral authentication using a global public key. If card-specific authentication keys are required, the procedure for the storage and authentication of card-specific public keys described in Section 4.11.3 is necessary.

As with symmetric authentication, the terminal generates a random number and sends this to the smart card. The card decrypts the random number using the secret key¹⁵ and then sends the result back to the terminal. The terminal holds the global public key, and it uses this key to

¹⁵ Using encryption to generate signatures comes from the convention that with an asymmetric cryptographic algorithm, the secret key is always used for decryption and the public key for encryption. The convention of using the public key for encryption goes back to the origins of the RSA procedure. One of the ideas at that time was to allow the RSA procedure to be used by agents operating in hostile territory to encrypt information to be kept secret. All that is needed to allow agents' reports to be sent back to headquarters in encrypted form is the RSA algorithm and

encrypt the random number that it has received. If the result of this computation is the same as the random number that was previously sent to the card, the card has been authenticated by the terminal.

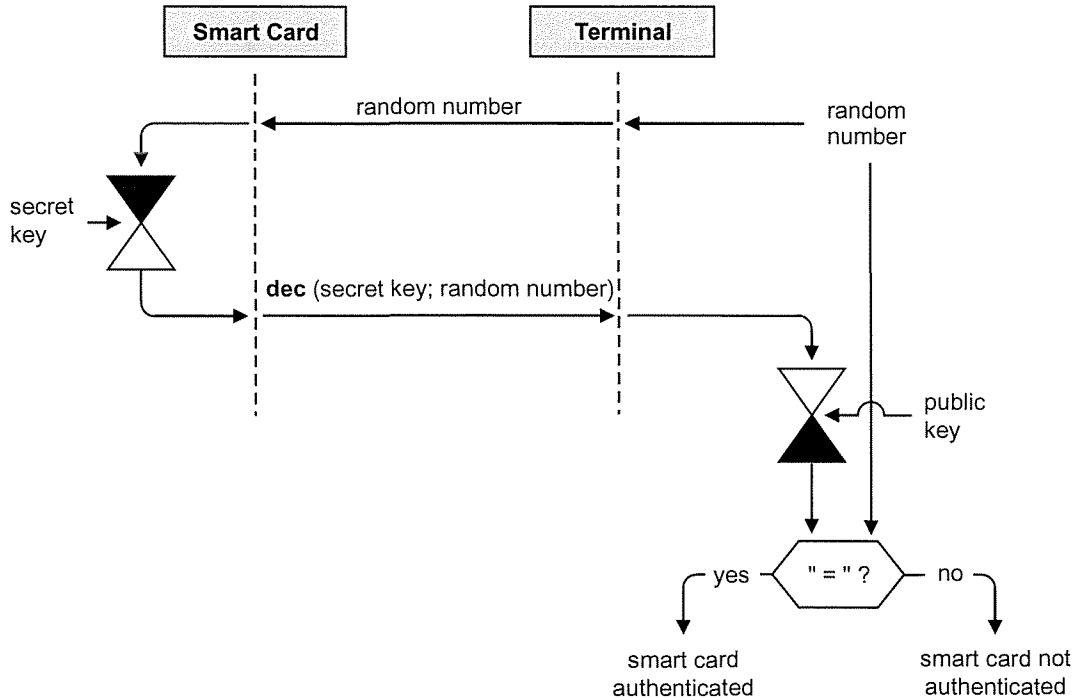


Figure 4.51 Operating principle of dynamic, asymmetric unilateral authentication of a smart card by a terminal

The basic features of a mutual authentication procedure for the smart card and the terminal are analogous to the unilateral procedure that has just been described. However, a mutual authentication requires a relatively long time, due to the large amount of data that must be exchanged and the time-consuming asymmetric encryption algorithm. Consequently, it is presently used very rarely.

4.12 DIGITAL SIGNATURES

Digital signatures, which are often referred to as 'electronic signatures', are used to establish the authenticity of electronically transmitted messages or electronic documents. Verification of the signature can be used to determine whether the message or document has been altered.

a public key. The messages can then be decrypted at headquarters in friendly territory using the private key. The main advantage of this arrangement is the ease of key distribution, since in principle agents can be given their keys without employing security measures. Even if a key becomes known, nobody would be able to decrypt messages that had been encrypted using the key, since this requires the private key. This initial strongly military application of the RSA algorithm forms the historical basis for the still valid convention that the public key is used for encryption and the private key for decryption

A signature has the property that it can be correctly produced by only one single individual, but it can be verified by anyone who receives the message – or at least, by any recipient who has previously seen the signature, or who has a copy available for comparison. This is also the essential characteristic of a digital signature. Only one person or one smart card can ‘sign’ a document, but everyone can verify whether the signature is genuine. Given this required characteristic, asymmetric cryptographic techniques represent the ideal starting point.

The message or document to be signed is usually at least several thousand bytes long. In order to keep the computation time for generating the cryptographic checksum within acceptable limits, the checksum is not computed over the entire data string. Instead, a hash value for the data string is first produced. Hash functions¹⁶ are, simply stated, one-way functions for data compression. This compression is not reversible, which means that the original data cannot be reconstructed from the compressed data. Since the computation of a hash value is very fast, hash functions are an ideal aid for computing digital signatures.

The term ‘digital signature’ is usually only used in connection with asymmetric cryptographic algorithms, since the separation of the public and private keys makes such algorithms very suitable for use with digital signatures. Nonetheless, ‘signatures’ based on symmetric cryptographic methods are often used in practice. However, with such signatures it is only possible to verify the authenticity of a document if the secret key used to generate the signature is known. Such a ‘signature’ is thus actually not a signature in the true sense of the word, but it is often referred to as such in practice. The term ‘digital’ is in this case omitted, to indicate the type of procedure used.

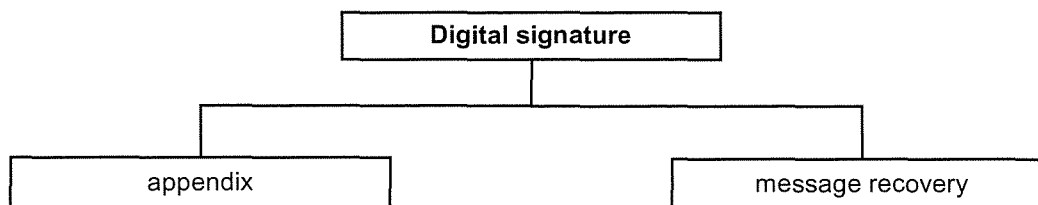


Figure 4.52 Classification of the two basic digital signature formats

From an informatics perspective, there are two different ways to attach a signature to a message. The first is a form of cryptographic checksum for a given data string, similar to a MAC (message authentication code), with the signature appended to the actual message (digital signature with appendix). This has the advantage that the message can be completely read without requiring prior verification of the signature. However, the drawback is that the size of the message is increased by the length of the signature, which can certainly be a consideration in the case of smart cards. This drawback can be avoided by using the second method for attaching a digital signature to a message, which is called ‘digital signature with message recovery’. In this method, the hash value of the actual message is first appended to the message, following which an input block for the digital signature algorithm is formed starting at the end of the resulting data string. This means that the digitally signed message is increased in size only by the length of the hash value, but it cannot be completely read until the digital signature has been verified.

¹⁶ See also Section 4.9, ‘Hash functions’

The procedure for generating a digital signature with appendix can be quite easily portrayed. First, a hash algorithm is used to form a hash value from the content of the message, which may for example be a file produced by any arbitrary word-processing program. This hash value is decrypted using an asymmetric cryptographic algorithm, such as RSA in the example shown in Figure 4.53. The result of this computation is the actual signature, which is appended to the message.

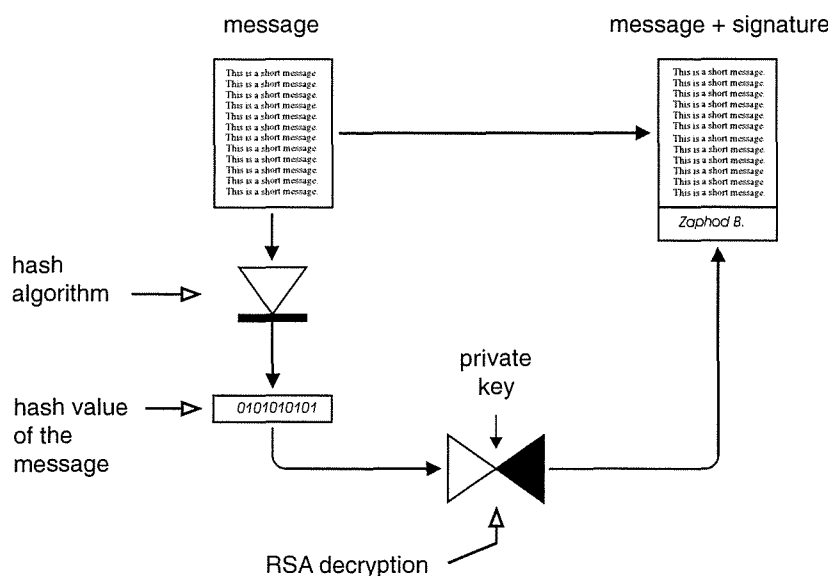


Figure 4.53 Signing a message with the RSA algorithm by appending the generated signature to the message (digital signature with appendix)

The signed message can now be sent via a non-secure path to the recipient. The recipient separates the signature from the message and then compresses the message using the same hash algorithm. The digital signature is encrypted using the public key of the RSA algorithm, and the result is compared with the result of the hash computation. If both values are the same, the message has not been altered while underway; otherwise, either the message or the signature has been altered during transmission. In the latter case, authenticity is no longer assured, and it cannot be assumed that the content of the message is unaltered.

The task of the smart card in this scenario is very simple. It stores at minimum the private RSA key, and it decrypts the hash value formed from the message, which means that it generates the signature. Everything else, such as generating the hash value or subsequently verifying the signature, can in principle be performed equally well by a PC.

Still, the ideal situation would be for the smart card to receive the message via its interface, compute the hash value and then send the signed message back to the terminal. Verification of the signature could also be performed by the smart card. This procedure is naturally no more secure than just computing the signature, but it is significantly more 'application-friendly'. This is because hash algorithms and RSA keys can be changed by simply exchanging the smart card, without any need to alter programs or data in the PC.

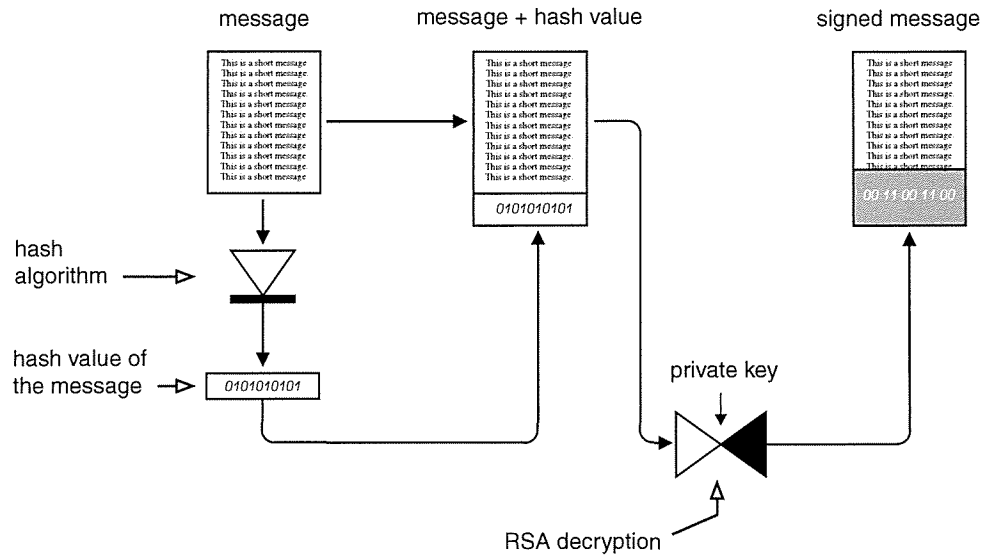


Figure 4.54 Signing a message with the RSA algorithm by incorporating the message and a hash value formed from the message in the signature (digital signature with message recovery)

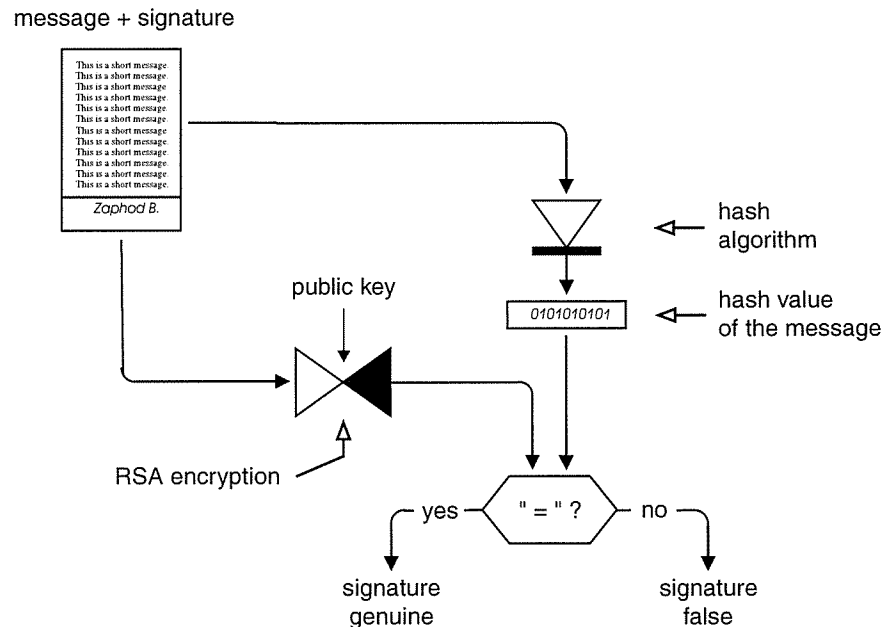


Figure 4.55 Verifying a message that has been signed using the RSA algorithm with the signature appended to the message (digital signature with appendix)

In these two examples, the keys used to generate and verify digital signatures are global, which means that they are the same for every smart card in a particular system. If a different arrangement should be used for security reasons, so that each card has its own key for digital signatures, a scheme such as that described in Section 4.11.3 must be used.

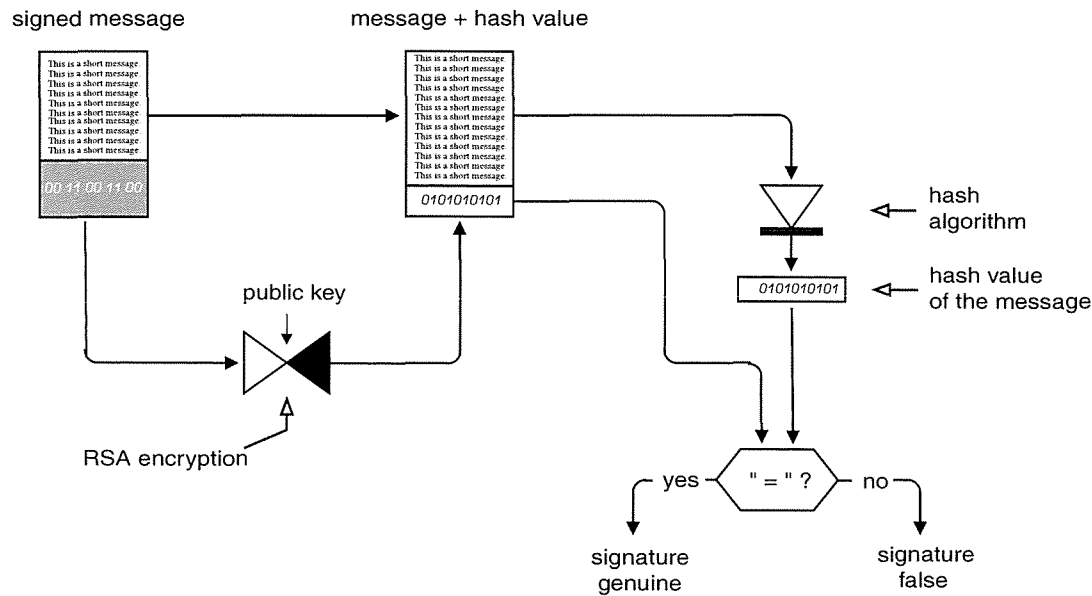


Figure 4.56 Verifying a message that has been signed using the RSA algorithm, in which a part of the message is used for the signature (digital signature with message recovery)

The RSA algorithm is not the only one that can be used to produce digital signatures. There is also a cryptographic procedure that has been specially developed for this application, namely the DSA (Digital Signature Algorithm). It was first proposed by the NIST (US National Institute of Standards and Technology) in 1991. Signatures can be both generated and verified using the DSA. In contrast to the RSA algorithm, it is largely designed such that it cannot be used for data encryption and decryption, although this has now been shown to not be true. Compared with the strong export restrictions applicable to the RSA algorithm and algorithms based on elliptic curves, such a feature would represent a major advantage for international use and/or export.

4.13 CERTIFICATES

With regard to the use of digital signatures, one is rather quickly confronted with a problem that must not be underestimated. Anyone who wants to verify the digital signature of a message needs the appropriate public key. However, the public key cannot be simply sent without any protection, since otherwise the recipient cannot verify the authenticity of the key. The public key must therefore be signed by a trustworthy body so that its authenticity can be verified. This body is called a certification authority (CA). The combination of a public key that has been signed by a certification authority, the accompanying digital signature and certain additional parameters is called a certificate.

There is also another body involved in this process, which is the trust center (TC). A trust center generates and manages certificates and associated blacklists, and it can optionally generate keys for digital signature cards. As a rule, a trust center also maintains a public

directory of certificates, so that anyone who wants to verify a signed message can request the associated signed public key from the center, for example via the Internet.

A certificate contains not only the signed public key, but also a large number of additional parameters and options, since it must be possible to verify the public key of a certificate without any further information. From this, it follows that among other things, the algorithms used to generate the hash value and the signature must be clearly specified. In principle, everyone who signs documents could specify his or her own certificate structure. Of course, this would make it impossible to exchange certificates. This would generally rob such certificates of their meaning, since exchangeability is an essential characteristic of a certificate.

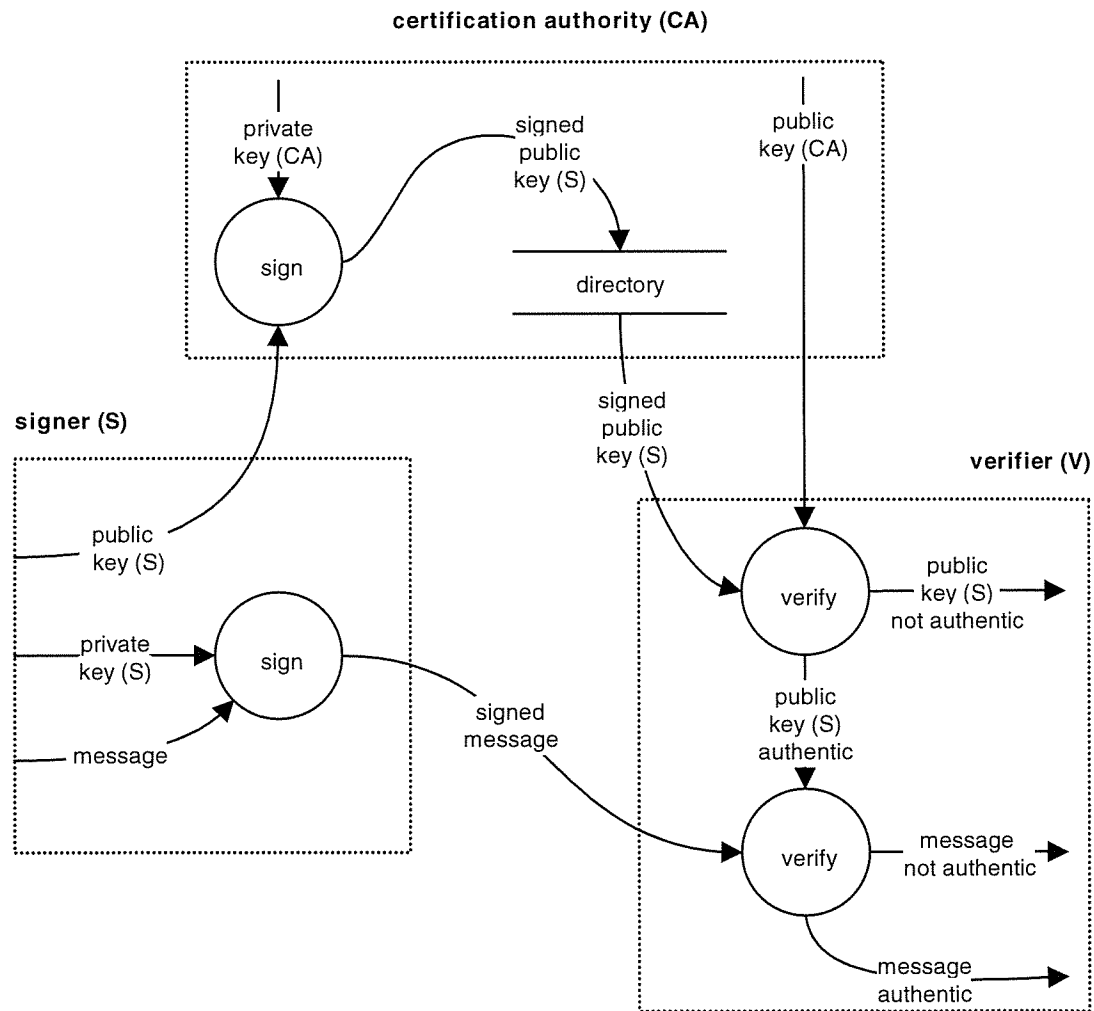


Figure 4.57 Data flow diagram of the basic processes for generating and verifying a transmitted message using a certificate. The certificate, which is generated by a certification authority, contains the public key of the signer and the signature of the certification authority

In order to insure that this sort of cooperation actually can take place, there are standards that specify the structure of certificates. The best known of the relevant standards is X.509,

which specifies the structure and coding of certificates. It has also entered the ranks of ISO/IEC standards as ISO/IEC 9594-8

The comprehensive X.509 standard is a framework in which the structure of certificates is defined in unambiguous terms. It forms the basis for many digital signature applications. Some examples are the Secure Socket Layer (SSL) Internet security mechanism and the Privacy Enhanced Mail (PEM), Secure Multipurpose Internet Mail Extensions (SMIME) and Secure Electronic Transaction (SET) applications.

ASN.1 is consistently used in X.509 to describe certificates, and the widely known TLV coding scheme is used in accordance with the Distinguished Encoding Rules (DER) for the actual coding.¹⁷ Some of the objects that may be included in a certificate are listed in Table 4.24. A brief introduction and summary of the subject of X.509 certificates can be found in a paper by Peter Gutmann [Gutmann 98b].

Table 4.24 Typical content of an X.509 certificate

Data element and X.509 designation	Explanation
Version	Identifies the version of X.509 that defines the data elements of this certificate. This is usually version 3.
Serial number	The serial number of the certificate. This must be issued by the issuer of the certificate, so that it is unique.
Signature algorithm identifier	Identifies the cryptographic algorithm used for the digital signature.
Issuer name	The name of the issuer of the certificate. The spelling of this name is unique, according to the X.500 standard.
Term of validity	The period for which the certificate is valid.
Subject name	The name of the entity whose public key should be recognized as authentic based on this certificate. According to the X.500 standard, the spelling of this name is unique
Public key	The subject entity's public key, which should be recognized as authentic based on this certificate.
Signature	The digital signature formed from the data of the certificate.

Many optional data fields for a wide variety of applications are defined in the X.509 standard. For example, it is easily possible to include several public keys in a single certificate and have them signed by different certification agencies. This can result in certificates that contain several kilobytes of data, which can cause problems if a smart card is to be used to verify the certificate. Of course, this scheme can also be used to generate items such as complementary certificates and tree-structured certificate hierarchies. A typical X.509 certificate in a smart card normally has a size of approximately 1 kB.

¹⁷ See also Section 4.1, 'Structuring Data'

