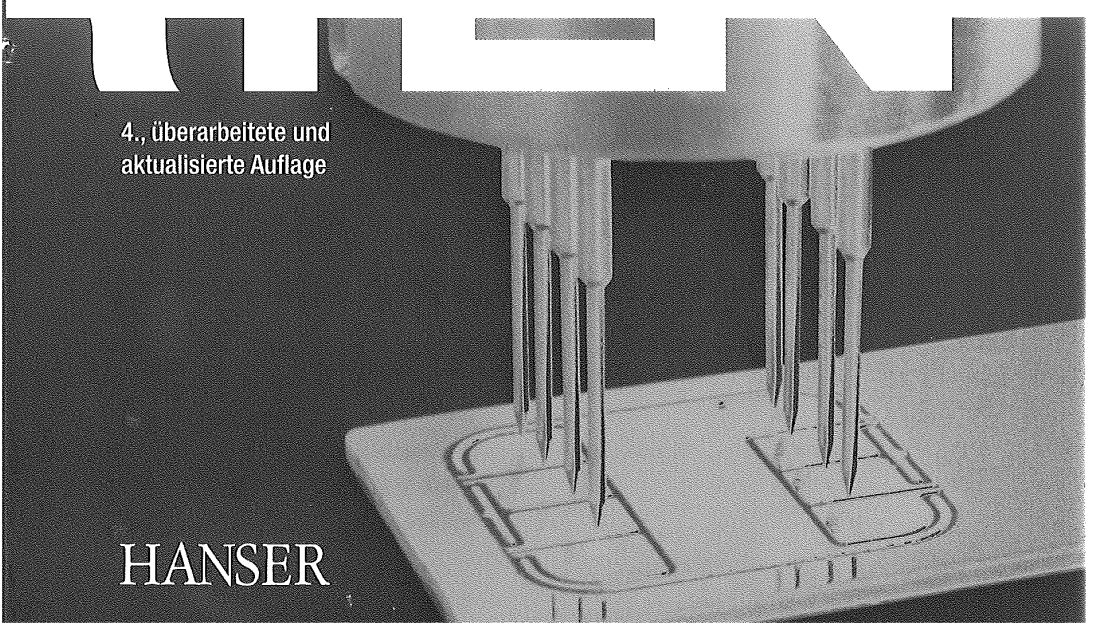
Website zum Buch

Wolfgang Rankl
Wolfgang Effing

Handbuch der Chipkarten

Aufbau – Funktionsweise – Einsatz von Smart Cards



4., überarbeitete und aktualisierte Auflage

HANSER

Rankl/Effing

Handbuch der Chipkarten

Wolfgang Rankl/Wolfgang Effing

Handbuch der Chipkarten

Aufbau – Funktionsweise – Einsatz von Smart Cards

4., überarbeitete und aktualisierte Auflage

HANSER

www.hanser.de

Die Deutsche Bibliothek – CIP-Einheitsaufnahme

Ein Titeldatensatz für diese Publikation
ist bei Der Deutschen Bibliothek erhältlich

Lektorat: Margarete Metzger
Herstellung: Irene Weilhart
Copy editing: Manfred Sommer
Satz: Druckhaus Thomas Müntzer, Bad Langensalza
Datenbelichtung, Druck und Bindung: Kösel, Kempten
Printed in Germany

ISBN 3-446-22036-4

Vorwort zur vierten Auflage

Verehrte Leser,

das Handbuch der Chipkarten liegt nunmehr in der vierten Auflage vor. Sie wurde gegenüber der letzten Auflage erheblich erweitert und durchgehend dem aktuellen Stand der Technik angepasst. Wir versuchen in diesem Buch die gesamte Chipkarten-Technologie abzudecken, wobei wir den Begriff „Technologie" bewusst sehr weit gefasst haben.

Wie schon in den vorherigen Auflagen haben wir unser Motto „lieber einen Satz zuviel als ein Wort zuwenig" beibehalten und das immer umfangreicher werdende Thema so detailliert wie möglich erläutert. Es sind auch weitere Beispiele und zusätzliche Bilder und Fotos zur Erklärung hinzugekommen, um die immer komplexer werdenden Zusammenhänge besser darstellen zu können. Ein stark erweitertes Glossar mit vielen Querverweisen deckt alle wesentlichen Begriffe im Umfeld von Karten ab und bietet sich in vielen Fällen als Schnelleinstieg in ein Thema an. Diese Ergänzungen, Erweiterungen und Verbesserungen haben dazu geführt, dass dieses Buch nunmehr einen Umfang von 353781 Wörtern in 61626 Zeilen besitzt. Die erste Auflage aus dem Jahr 1995 umfasste 97145 Wörter.

An dieser Stelle sei ein kurzer Vergleich aufgeführt. Moderne Chipkarten-Betriebssysteme haben derzeit einen Umfang von 120 000 Zeilen Quellcode, das entspricht ungefähr dem Umfang von zwei Büchern dieser Auflage. Selbst ohne Programmierkenntnisse ist daraus deutlich zu erkennen, welch enormer Entwicklungsstand hier bereits erreicht wurde.

Die kleinen bunten Plastikkarten mit dem Halbleiterchip breiten sich von ihren Ursprungsländern Deutschland und Frankreich über die ganze Welt aus. Diesem Siegeszug wird auch in den nächsten Jahren keine andere Technologie Einhalt gebieten, zudem wir uns immer noch am Anfang der Entwicklung befinden, und weder ein Ende noch eine Konsolidierung abzusehen ist.

Während die Chipkarten-Technologie in großen und schnellen Schritten voranschreitet, versuchen wir mit den alle zwei bis drei Jahre erscheinenden Auflagen des Handbuchs der Chipkarten einher zu gehen. Es repräsentiert das aktuelle technische Wissen und wir haben in Bereichen, die derzeit einer schnellen Weiterentwicklung unterliegen, mögliche Evolutionspfade aufgezeigt. Sollten manche Dinge zu einem späteren Zeitpunkt in einem etwas anderen Lichte erscheinen, so können wir nur anmerken, dass niemand die genauen Pfade der Zukunft kennt. Trotzdem oder gerade deswegen freuen wir uns über jeden Kommentar, jede Anregung und jeden Verbesserungsvorschlag, damit dieses Buch auch weiterhin das Thema Chipkarten so vollständig wie möglich behandeln kann. Wir möchten uns an dieser Stelle ausdrücklich bei den vielen aufmerksamen und engagierten Lesern bedanken, die uns auf Unklarheiten, missverständliche Passagen und Fehler aufmerksam gemacht haben. Auch für diese Auflage wird es wieder eine Errata-Liste bei [Hanser] und [Rankl] geben.

Bei den vielen Freunden und Kollegen, die uns immer wieder wertvolle und manchmal auch etwas unbequeme Hinweise gegeben haben, um dieses Buch vollständiger und besser zu machen, möchten wir uns ebenfalls bedanken. Unser besonderer Dank gilt dabei: Hermann Altschäfl, Peter van Elst, Klaus Finkenzeller, Thomas Graßl, Michael Schnellinger, Harald Vater, Dieter Weiß und natürlich Margarete Metzger und Irene Weilhart vom Carl Hanser Verlag für die gute Unterstützung in allen verlags- und buchtechnischen Belangen.

München, im Juni 2002

Wolfgang Rankl
[Rankl@gmx.net], [Rankl]

Wolfgang Effing
[WEffing@gmx.net]

Inhalt im Überblick

Inhaltsverzeichnis

Symbole/Notationen

- Das niedrigstwertige Bit wird analog ISO einheitlich mit der Nummer 1 bezeichnet und nicht mit der Nummer 0.
- Die Angabe „Byte" ist analog dem üblichen Sprachgebrauch eine Sequenz von 8 Bit, ist also dem in der internationalen Normung oft verwendeten Begriff „Oktett" gleichzusetzen.
- Die Längenangaben von Daten oder Objekten sowie alle zählbaren Größen sind, analog der in der Chipkarten-Normung üblichen Notation, in dezimaler Schreibweise dargestellt. Alle anderen Werte sind in der Regel als Hexadezimalzahlen aufgeführt und auch als solche gekennzeichnet.
- Die Präfixe „Kilo" und „Mega" haben, wie im IT-Bereich üblich, im Zusammenhang mit Daten- oder Speichergrößen die Werte 1024 (= 2^{10}) und 1 048 576 (= 2^{20}).
- Binäre Werte sind kontextabhängig nicht immer explizit gekennzeichnet.
- Kommandos im Zusammenhang mit Chipkarten sind in Großbuchstaben geschrieben (z. B.: SELECT FILE).

Zeichen- und Zahlendarstellung

42	dezimaler Wert
'00'	hexadezimaler Wert
°0°, °1°	binärer Wert
"ABC"	ASCII-Wert
Bn	Byte mit der Nummer n (z. B.: B1)
bn	Bit mit der Nummer n (z. B.: b2)
Dn	Stelle (Digit) mit der Nummer n (z. B.: D3)

logische Funktionen

$\|\|$	Verkettung (von Datenelementen oder Objekten)
$\oplus$	logische XOR-Verknüpfung
$\wedge$	logische UND-Verknüpfung
$\vee$	logische ODER-Verknüpfung
$a \in M$	a ist ein Element der Menge M
$a \notin M$	a ist kein Element der Menge M
{a, b, c}	Menge der Elemente a, b, c

kryptografische Funktionen

$enc_{X\ n}$ (K; D)	Verschlüsselung mit dem Algorithmus X, der Schlüssellänge n Bit, dem Schlüssel K und den Daten D (z. B.: $enc_{DES\ 56}$ ('1 ... 0'; 42)).
$dec_{X\ n}$ (K; D)	Entschlüsselung mit dem Algorithmus X, der Schlüssellänge n Bit, dem Schlüssel K und den Daten D (z. B.: $dec_{IDEA\ 128}$ ('1 ... 0'; 42).
S := $sign_{X\ n}$ (K; D)	Erzeugung der Signatur S mit dem Algorithmus X, der Schlüssellänge n Bit, dem Schlüssel K und den Daten D (z. B.: $sign_{RSA\ 512}$ ('1 ... 0'; "Wolf")).
R := $verify_{X\ n}$ (K; S)	Prüfung der Signatur S mit dem Algorithmus X, der Schlüssellänge n Bit und dem Schlüssel K (z. B.: $verify_{RSA\ 512}$ ('1 ... 9'; 42)). R ist das Ergebnis dieser Operation, das richtig oder falsch sein kann.

Verweise

siehe „ ... "	Dies ist ein Querverweis auf eine andere Stelle innerhalb des Buches.
siehe auch „ ... "	Dies ist ein Querverweis auf eine andere Stelle innerhalb des Buches an der sich weitere Informationen zu dem Thema befinden.
[...]	Dies ist ein Querverweis auf im Anhang genannte World-Wide-Web-Seiten.
[X Y]	Dies ist ein Querverweis auf im Anhang genannte weiterführende Literatur. Es gilt X ∈ {Familienname des erstgenannten Autors} und Y ∈ {auf die Zehnerstellen gekürzte Jahreszahl des Erscheinungsdatums}.

Programmcode

Der in diesem Buch verwendete Programmcode orientiert sich in Syntax und Semantik an den gängigen BASIC-Dialekten. Zur Vereinfachung des Verständnisses ist es jedoch zusätzlich erlaubt, innerhalb des Programmcodes Erklärungen in natürlicher Sprache einzufügen. Damit wird es für den Leser zwar einfacher, den dargestellten Programmcode zu verstehen, einer automatischen Übersetzung in Maschinencode ist dadurch allerdings der Weg versperrt. Die deutlich erhöhte Lesbarkeit rechtfertigt aber diesen Kompromiss in jedem Falle.

:=	Zuweisungsoperator
=, <, >, <>, <=, >=	Vergleichsoperatoren
+, -, *, /	Rechenoperatoren
NOT	logisches Nicht
AND	logisches Und
OR	logisches Oder
||	Verkettungsoperator (z. B.: von zwei Bytestrings)
_	Zeilenende innerhalb mehrzeiliger Anweisungen
// ...	Kommentar
IO_Buffer	Variable (in kursiver Schrift gesetzt)
Label:	Sprungziel/Aufrufziel (in fetter Schrift gesetzt)
GOTO ...	Sprung
CALL ...	Funktionsaufruf
RETURN	Funktionsrücksprung
IF ... THEN ...	Entscheidung, Variante 1
IF ... THEN ... ELSE ...	Entscheidung, Variante 2
SEARCH (...)	Suchen in einer Liste, Beschreibung als Text in Klammern
STATUS (...)	Abfrage des Ausführungsergebnisses bei einem vorangegangenem Funktionsaufruf
STOP	Beendigung eines Ablaufs
LENGTH (...)	Berechnung von Längen
EXIST	Test auf Vorhandensein (z. B.: eines Objekts oder Datenelements)
WITH ...	Beginn der Festlegung einer Variablen/eines Objekts als Referenz
END WITH	Ende der Festlegung einer Variablen/eines Objekts als Referenz

Abkürzungen

3DES	Triple-DES *(siehe Glossar)*
3GPP	third generation partnership project *(siehe Glossar)*
3GPP2	third generation partnership project 2 *(siehe Glossar)*
A3, A5, A8	GSM Algorithm 3, 5, 8 *(siehe Glossar)*
AAM	application abstract machine
ABA	American Bankers Association
ABS	Acrylnitril-Butadien-Styrol
AC	access conditions *(siehe Glossar)*
ACD	access control descriptor
ACK	acknowledge
ACM	Accumulated Call Meter
ADF	application dedicated file
ADN	abbreviated dialling number
AES	Advanced Encryption Standard *(siehe Glossar)*
AFI	Application family identifier
AFNOR	Association Française de Normalisation *(siehe Glossar)*
AGE	Autobahngebührenerfassung
AGE	automatische Gebührenerfassung
AID	application identifier *(siehe Glossar)*
AM	access mode
Amd.	Amendment
AMPS	Advanced Mobile Phone Service *(siehe Glossar)*
AND	logische UND-Verknüpfung
ANSI	American National Standards Institute *(siehe Glossar)*
AoC	Advice of Charge
AODF	authentication object directory file
APACS	Association for Payment Clearing Services
APDU	application protocol data unit *(siehe Glossar)*
A-PET	amorphes Polyethylenterephthalat
API	application programming interface *(siehe Glossar)*
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 *(siehe Glossar)*
ASN.1	abstract syntax notation one *(siehe Glossar)*
AT	attention
ATM	automated teller machine
ATQA	Answer to request, type A
ATQB	Answer to request, type B
ATR	answer to reset *(siehe Glossar)*
ATS	Answer to Select
ATTRIB	PICC selection command, type B
AUX	auxiliary
B2A	business to administration *(siehe Glossar)*
B2B	business to business *(siehe Glossar)*
B2C	business to consumer *(siehe Glossar)*

BASIC	beginners all purpose symbolic instruction code
BCD	binary coded digit
Bellcore	Bell Communications Research Laboratories
BER	basic encoding rules *(siehe Glossar)*
BER-TLV	basic encoding rules tag, length, value
BEZ	Börsenevidenzzentrale
BGT	block guard time
BIN	bank identification number
bit	binary digit
BPF	basic processor functions
BPSK	Binary phase shift keying *(siehe Glossar)*
BS	base station
BWT	block waiting time
CA	certificate authority *(siehe Glossar)*
CAD	chip accepting device *(siehe Glossar)*
CAFE	conditional access for Europe (EU-Projekt)
CAMEL	Customized Applications for Mobile Enhanced Logic
CAP	card application *(siehe Glossar)*
C-APDU	command-APDU *(siehe Glossar)*
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 *(siehe Glossar)*
CCD	card coupling device
CCD	charge coupled device
CCITT	Comité Consultatif International Télégraphique et Téléphonique (jetzt ITU) *(siehe Glossar)*
CCR	chip card reader
CCS	cryptographic checksum *(siehe Glossar)*
CD	committee draft
CDF	certificate directory file
CDM	card dispensing machine
CDMA	Code Division Multiple Access *(siehe Glossar)*
CEN	Comité Européen de Normalisation *(siehe Glossar)*
CENELEC	Comité Européen de Normalisation Eléctrotechnique (Europäisches Komitee für elektrotechnische Standardisierung)
CEPS	common electronic purse specifications, früher: common european purse system *(siehe Glossar)*
CEPT	Conférence Européenne des Postes et Télécommunications *(siehe Glossar)*
CFB	cipher feedback
CGI	Common Gateway Interface
CHV	card holder 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 *(siehe Glossar)*

CMOS	complementary metal oxide semiconductor
CMS	card management system
COS	chip operating system *(siehe Glossar)*
COT	chip-on-tape *(siehe Glossar)*
CRC	cyclic redundancy check *(siehe Glossar)*
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 *(siehe Glossar)*
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 *(siehe Glossar)*
DAM	draft amendment
D-AMPS	Digital Advanced Mobile Phone Service *(siehe Glossar)*
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 *(siehe Glossar)*
DECT	digital enhanced cordless telecommunications (früher: digital european cordless telecommunications) *(siehe Glossar)*
DER	distinguished encoding rules *(siehe Glossar)*
DES	data encryption standard *(siehe Glossar)*
DF	dedicated file (oft auch: directory file) *(siehe Glossar)*
DFA	differential fault analysis *(siehe Glossar)*
DFÜ	Datenfernübertragung
DIL	dual inline
DIN	Deutsche Industrienorm
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 *(siehe Glossar)*
dpi	dots per inch

DR	Divisor receive (PCD to PICC)
DRAM	dynamic random access memory *(siehe Glossar)*
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	Datenträgeraustausch
DTD	Document Type Definition
DTMF	dual tone multiple frequency
DVD	digital versatile disc
DVS	Dateiverwaltungssystem
E	End of communication, type A
E/A	Eingabe/Ausgabe
EBCDIC	extended binary coded decimal interchange code
EC	elliptic curve
ec	eurocheque
ECB	electronic code book
ECBS	European Committee for Banking Standards *(siehe Glossar)*
ECC	elliptic curve cryptosystems *(siehe Glossar)*
ECC	error correction code *(siehe Glossar)*
ECDSA	elliptic curve DSA
ECML	Electronic Commerce Modelling Language
ECTEL	European Telecom Equipment and Systems Industry
EDC	error detection code *(siehe Glossar)*
EDGE	enhanced data rates for GSM and TDMA evolution *(siehe Glossar)*
EDI	electronic data interchange
EDIFACT	electronic data interchange for administration, commerce and transport
EEPROM, E^2PROM	electrical erasable programmable read only memory *(siehe Glossar)*
EF	elementary file *(siehe Glossar)*
EFF	electronic frontier foundation
EFI	EF internal
EFTPOS	electronic found transfer at point of sale
EFW	EF working
EGT	Extra guard time, type B
EMV	Europay, MasterCard, Visa *(siehe Glossar)*
EOF	End of frame, type B
EPROM	erasable programmable read only memory *(siehe Glossar)*
ESD	electrostatic discharge
ESPRIT	European Strategic Programme of Research and Development in Information Technology (EU Projekt)
ETS	European Telecommunication Standard *(siehe Glossar)*
ETSI	European Telecommunications Standards Institute *(siehe Glossar)*
etu	elementary time unit *(siehe Glossar)*
f	folgende Seite
FAQ	frequently asked questions
FAR	false acceptance rate
FAT	file allocation table *(siehe Glossar)*
FBZ	Fehlbedienungszähler *(siehe Glossar)*
fC	Frequency of operating field (carrier frequency)
FCB	file control block
FCC	Federal Communication Commission

FCFS	first come first serve
FCI	file controll information
FCOS	flip chip on substrate
FCP	file control parameters
FD/CDMA	frequency division/code division multiple access *(siehe Glossar)*
FDMA	Frequency Division Multiple Access *(siehe Glossar)*
FDN	fixed dialling number
FDT	Frame delay time, type A
FEAL	fast data encipherment algorithm
FET	field effect transistor
ff	folgende Seiten
FID	file identifier *(siehe Glossar)*
FIFO	first in last out
FINEID	Finnish Electronic Identification Card
FIPS	Federal Information Processing Standard *(siehe Glossar)*
FMD	file management data
FO	Frame option
FPGA	field programmable gate array *(siehe Glossar)*
FPLMTS	future public land mobile telecommunication service *(siehe Glossar)*
FRAM	ferroelectric random access memory *(siehe Glossar)*
FRR	false rejection rate
FS	file system
fS	Frequency of sub carrier 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
GF	Galois-Felder
GGSN	Gateway GPRS Support Node
ggt	größter gemeinsamer Teiler
GND	ground
GP	Global Platform *(siehe Glossar)*
GPL	GNU public license
GPRS	General Packet Radio System *(siehe Glossar)*
GPS	global positioning system
GSM	Global System for Mobile Communications, früher: Groupe Special Mobile *(siehe Glossar)*
GTS	GSM Technical Specification
GUI	graphical user interface
HAL	Hardware Abstraction Layer *(siehe Glossar)*
HBCI	Home Banking Computer Interface *(siehe Glossar)*
HiCo	high coercivity
HLTA	Halt command, Typ A
HLTB	Halt command, Typ B
HSCSD	High Speed Circuit Switched Data
HSM	hardware security module

HSM	high security module
HSM	Hochsicherheitsmodul
HSM	host security module
HTML	hyper text markup language
HTTP	hyper text transfer protocol
HV	Härte Vickers
HW	hardware
I/O	input/output
I^2C	inter-integrated circuit
IATA	International Air Transport Association
IBAN	international bank account number
I-block	Information block
ICC	integrated circuit card *(siehe Glossar)*
ID	identifier
IDEA	international data encryption algorithm
IEC	International Electrotechnical Commission *(siehe Glossar)*
IEEE	Institute of Electrical and Electronics Engineers
IEP	inter-sector electronic purse
IFD	interface device *(siehe Glossar)*
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 *(siehe Glossar)*
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 *(siehe Glossar)*
ISF	internal secret file
ISIM	IP security identity module
ISO	International Organization for Standardization *(siehe Glossar)*
IT	Informationstechnik
ITSEC	Information Technology Security Evaluation Criteria *(siehe Glossar)*
ITU	International Telecommunicatios Union *(siehe Glossar)*
IuKDG	Informations- und Kommunikations-Gesetz
IV	Initialisierungsvektor
IVU	in vehicle unit
J	Jahr
J2ME	Java 2 Micro Edition
JCF	Java Card Forum *(siehe Glossar)*
JCRE	Java Card Runtime Environment *(siehe Glossar)*
JCVM	Java Card Virtual Machine *(siehe Glossar)*
JDK	Java Development Kit *(siehe Glossar)*
JECF	Java electronic commerce framework
JIT	just in time

JTC1	Joint Technical Commitee One
JVM	Java Virtual Machine
K	key
Kc	ciphering key
KD	key derived
KFPC	key fault presentation counter
Ki	individual key
KID	key identifier
KM	key master
KS	key session
KVK	Krankenversichertenkarte
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 dialled
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	Monat
MAC	Message Authentication Code/Datensicherungscode *(siehe Glossar)*
MAOS	multi application operating system
MBL	Maximum buffer length
MBLI	Maximum buffer length index
ME	mobile equipment
MEL	Multos executable language
MExE	mobile station execution environment *(siehe Glossar)*
MF	master file *(siehe Glossar)*
MFC	multi function card, multifunktionale Chipkarte
MIME	multipurpose internet mail extensions
MIPS	million instructions per second
MKT	Multifunktionales Kartenterminal *(siehe Glossar)*
MLI	multiple laser image
MM	moduliertes Merkmal
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 *(siehe Glossar)*
MS	mobile station

msb	most significant bit
MSB	most significant byte
MSE	MANAGE SECURITY ENVIRONMENT
MTBF	mean time between failure
MUSCLE	Movement for the Use of Smart Cards in a Linux Environment
NAD	node address
NAK	Negativ acknowledgement
NBS	US National Bureau of Standards *(siehe Glossar)*
NCSC	National Computer Security Center *(siehe Glossar)*
NDA	non disclosure agreement
NIST	US National Institute of Standards and Technology *(siehe Glossar)*
nok	nicht ok
NPU	numeric processing unit *(siehe Glossar)*
NRZ	non return to zero
NSA	US National Security Agency *(siehe Glossar)*
NU	not used
NVB	Number of valid bits
OBU	on board unit
ODF	object directory file
OFB	output feedback
OID	object identifier
OOK	On/off keying
OP	Open Platform *(siehe Glossar)*
OR	logische ODER-Verknüpfung
OS	operating system
OSI	Open Systems Interconnections
OTA	Open Terminal Architecture
OTA	over the air *(siehe Glossar)*
OTASS	over the air SIM services
OTP	one time password
OTP	one time programmable
OTP	open trading protocol
OVI	optically variable ink
ÖPNV	öffentlicher Personennahverkehr
P1, P2, P3	Parameter 1, 2, 3
PA	power analysis
PB	procedure byte
PC	personal computer
PC	Polycarbonat
PC/SC	personal computer/smart card *(siehe Glossar)*
PCB	protocol control byte
PCD	proximity coupling device *(siehe Glossar)*
PCMCIA	Personal Computer Memory Card International Association
PCN	personal communication networks
PCS	personal communication system
PDA	personal digital assistant
PES	proposed encryption standard
PET	Polyethylenterephthalat
PETP	teilkristallines Polyethylenterephthalat

PGP	Pretty Good Privacy
PICC	proximity ICC *(siehe Glossar)*
PIN	personal identification number
PIX	proprietary application identifier extension
PKCS	public key cryptography standards *(siehe Glossar)*
PKI	public key infrastructure *(siehe Glossar)*
PLL	phase locked loop
PLMN	Public Land Mobile Network *(siehe Glossar)*
PM	Personenmonat
POS	point of sale *(siehe Glossar)*
POZ	POS ohne Zahlungsgarantie
PP	Protection Profil *(siehe Glossar)*
PPM	pulse position modulation
PPS	Produktionsplanungs- und steuerung
PPS	protocol parameter selection
prEN	pre Norme Européenne
prETS	pre European Telecommunication Standard
PrKDF	private key directory file
PRNG	pseudo random number generator *(siehe Glossar)*
PROM	programmable read only memory
PSAM	purchase secure application module
PSK	phase shift keying
PSO	PERFORM SECURITY OPERATION
PSTN	public switched telephone network *(siehe Glossar)*
PTS	protocol type selection
PTT	Postes Télégraphes et Téléphones
Pub	publication
PUK	personal unblocking key *(siehe Glossar)*
PuKDF	public key directory file
PUPI	Pseudo-unique PICC identifier
PVC	Polyvinylchlorid
PWM	pulse width modulation
RAM	random access memory *(siehe Glossar)*
R-APDU	response-APDU *(siehe Glossar)*
RATS	Request to answer to select
REJ	reject
REQA	Request command, type A
REQB	Request command, type B
RES	resynchronisation
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-Projekt) 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 *(siehe Glossar)*
RS	Reed-Solomon

RSA	Rivest, Shamir und Adleman Kryptoalgorithmus
RTE	runtime environment
R-UIM	removable user identity module *(siehe Glossar)*
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 *(siehe Glossar)*
SAT	SIM Application Toolkit *(siehe Glossar)*
SC	security conditions
SC	smart card
SCC	smart card controller
SCMS	smart card management system
SCOPE	smart card open platform environment *(siehe Glossar)*
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 *(siehe Glossar)*
SE	security environment *(siehe Glossar)*
SECCOS	Secure Chip Card Operating System *(siehe Glossar)*
SEIS	secured electronic information in society
SEL	Select code
SELECT	Select command
SEMPER	Secure Electronic Marketplace for Europe (EU-Projekt)
SEPP	secure electronic payment protocol
SET	secure electronic transaction *(siehe Glossar)*
SFGI	Start-up frame guard time integer
SFGT	Start-up frame guard time
SFI	short file identifier *(siehe Glossar)*
SGSN	Serving GPRS Support Node
S-HTTP	secure hyper text transfer protocol
SigG	Signaturgesetz *(siehe Glossar)*
SigV	Signaturverordnung *(siehe Glossar)*
SIM	subscriber identity module *(siehe Glossar)*
SIMEG	subscriber identity module expert group *(siehe Glossar)*
SKDF	secret key directory file
SM	secure messaging
SM	security mechanism
SMD	surface mounted device *(siehe Glossar)*
SMG9	Special Mobile Group 9 *(siehe Glossar)*
SMIME	secure multipurpose internet mail extensions
SMS	short messages service *(siehe Glossar)*
SMSC	Short Message Service Center
SMS-PP	Short Message Service Point to Point
SOF	Start of frame
SPA	simple power analysis *(siehe Glossar)*

SQL	structured query language
SQUID	superconducting quantum interference device
SRAM	static random access memory *(siehe Glossar)*
SRES	signed response
SS	sublementary service
SSC	send sequence counter
SSL	secure socket layer
SSO	Single-Sign-On *(siehe Glossar)*
STARCOS	Smart Card Chip Operating System (Produkt von G+D)
STC	sub technical committee
STK	SIM Application Toolkit *(siehe Glossar)*
STT	secure transaction technology
SVC	stored value card (Produkt von 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	Transaktionsnummer *(siehe Glossar)*
TAR	Toolkit Application Reference
tbd	to be defined
TC	Trustcenter *(siehe Glossar)*
TC	Technical Commitee
TC	thermochrom
TCOS	Telesec Card Operating System
TCP	transport control protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
TCSEC	Trusted Computer System Evaluation Criteria *(siehe Glossar)*
TD/CDMA	time division/code division multiple access *(siehe Glossar)*
TDES	Triple-DES *(siehe Glossar)*
TDMA	Time Division Multiple Access *(siehe Glossar)*
TETRA	Trans-European Trunced Radio *(siehe Glossar)*
TLS	Transport Layer Security
TLV	tag, length, value *(siehe Glossar)*
TMSI	temporary mobile subscriber identity
TOE	target of evaluation *(siehe Glossar)*
TPDU	transmission protocol data unit *(siehe Glossar)*
TRNG	true random number generator *(siehe Glossar)*
TS	technical specification
TTCN	the tree and tabular combined notation
TTL	terminal transport layer
TTL	transistor transistor logic
TTP	Trusted-Third-Party *(siehe Glossar)*
UART	universal asynchronous receiver transmitter *(siehe Glossar)*
UATK	UIM Application Toolkit
UCS	universal character set *(siehe Glossar)*
UI	User Interface
UICC	universal integrated circuit card *(siehe Glossar)*
UID	Unique identifier

UIM	user identity module *(siehe Glossar)*
UML	unified modeling language *(siehe Glossar)*
UMTS	Universal Mobile Telecommunication System *(siehe Glossar)*
URL	uniform resource locator *(siehe Glossar)*
USAT	USIM Application Toolkit *(siehe Glossar)*
USB	universal serial bus
USIM	universal subscriber identity module *(siehe Glossar)*
USSD	unstructured supplementary services data
UTF	UCS transformation format
UTRAN	UMTS Radio Access Network
VAS	value added services *(siehe Glossar)*
Vcc	Versorgungsspannung
VCD	vicinity coupling device
VEE	Visa Easy Entry *(siehe Glossar)*
VKNR	Versichertenkartennummer
VLSI	very large scale integration
VM	virtual machine *(siehe Glossar)*
VOP	Visa Open Platform *(siehe Glossar)*
Vpp	Programmierspannung
VSI	vertical system integration
W3C	World Wide Web Consortium
WAE	Wireless Application Environment
WAN	wide area network
WAP	Wireless Application Protocol *(siehe Glossar)*
WCDMA	wideband code division multiple access *(siehe Glossar)*
WDP	Wireless Datagram Protocol
WfSC	Windows for Smart Cards
WG	working group
WIG	Wireless Internet Gateway
WIM	wireless identification module *(siehe Glossar)*
WML	Wireless Markup Language *(siehe Glossar)*
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	Wake up command, type A
WUPB	Wake up command, type B
WWW	World Wide Web *(siehe Glossar)*
XML	Extensible Markup Language *(siehe Glossar)*
XOR	logische exclusive ODER-Verknüpfung
ZKA	Zentraler Kreditausschuss *(siehe Glossar)*

1 Einleitung

Dieses Buch wendet sich an Studenten, Ingenieure und technisch Interessierte, die mehr über das Thema Chipkartentechnik wissen möchten. Es versucht, dieses sehr breite Gebiet so vollständig wie möglich abzudecken, um so dem Leser einen Überblick über die Grundlagen und den aktuellen Stand der Technik zu geben.

Wir haben großen Wert darauf gelegt, die Thematik möglichst praxisnah darzustellen. Anhand der vielen Bilder, Tabellen und Bezüge zu realen Anwendungen von Chipkarten möchten wir dem Leser helfen, sich in die Thematik wesentlich schneller einzuarbeiten als bei einer streng wissenschaftlichen Darstellungsweise. Deshalb erhebt dieses Buch auch nicht den Anspruch auf wissenschaftliche Vollständigkeit, sondern auf eine praxisnahe Gebrauchsfähigkeit. Dies ist auch der Grund, warum die Erklärungen so wenig abstrakt wie möglich gehalten sind. Es war an vielen Stellen eine Entscheidung für wissenschaftliche Exaktheit oder für leichte Verständlichkeit zu treffen. Wir haben versucht, einen Mittelweg zu gehen, falls dies aber nicht möglich war, ging uns die Verständlichkeit des Themas immer vor.

Der Aufbau des Buches ist derart, dass es ganz normal von vorne nach hinten gelesen werden kann, da wir so weit wie möglich versucht haben, Vorwärtsbezüge zu vermeiden. Die einzelnen Kapitel sind in ihrem Aufbau und Inhalt so gestaltet, dass sie auch separat ohne Einbußen an Verständnis gelesen werden können. Das ausführliche Sachverzeichnis und Glossar machen das Handbuch der Chipkarten darüber hinaus zu einem praktischen Nachschlagewerk. Möchte man mehr über ein spezielles Gebiet wissen, dann helfen die Querverweise im Text und das kommentierte Normenverzeichnis, die entsprechende Stelle zu finden.

Leider haben sich, wie in so vielen Gebieten der Technik und des täglichen Lebens, auch in der Chipkartentechnik mittlerweile sehr viele Abkürzungen eingebürgert. Dies macht gerade für den Neuling den Einstieg besonders schwer. Auch hier haben wir wiederum versucht, so wenig wie möglich von den kryptischen und oft auch unlogischen Kurzformen Gebrauch zu machen. Allerdings musste in vielen Fällen ein Mittelweg zwischen dem international üblichen Fachjargon der Spezialisten und der allgemein verständlichen Umgangssprache der Laien gewählt werden. Falls uns dies nicht immer gelungen ist, so soll wenigstens das sehr umfangreiche Abkürzungsverzeichnis die (hoffentlich nur) anfängliche Mauer der Unverständlichkeit überwinden helfen. Ein umfangreiches Glossar am Ende des Buches erläutert die wichtigsten Fachbegriffe in prägnanter Form und ergänzt so das Abkürzungsverzeichnis.

Mit der mittlerweile unüberschaubaren Anzahl von Anglizismen im Umfeld von Chipkarten standen wir vor dem gleichen Problem wie mit den Abkürzungen. Auch hier haben wir versucht, so weit als möglich und sinnvoll die Klarheit einer einheitlichen Sprache

beizubehalten und nur bei denjenigen Begriffen für die es keine sinnvollen oder nur unübliche Übersetzungen gibt, die entsprechenden englischen Begriffe zu wählen. Gleichwohl haben wir an den entsprechenden Stellen die entsprechenden englischen oder deutschen Begriffe aufgeführt.

Ein wichtiges Merkmal von Chipkarten besteht darin, dass sie in ihren Eigenschaften stark auf internationalen Normen aufbauen. Dies ist aufgrund der meist zwingenden Interoperabilität auch von wesentlicher Bedeutung. Leider sind diese Normen oft schwierig zu verstehen und müssen an einigen kritischen Stellen sogar interpretiert werden. Manchmal können ausschließlich die Mitglieder der jeweiligen Normungsgruppe Auskunft über die eigentliche Intention von bestimmten Abschnitten Auskunft geben. Das Handbuch der Chipkarten versucht an diesen Stellen, den in der Chipkartenindustrie üblicherweise akzeptierten Weg aufzuzeigen. Trotzdem ist und bleiben die entsprechenden Normen die alleinige Referenz und sollten in solchen Fällen auch immer zum Studium herangezogen werden.

1.1 Geschichte der Chipkarten

Die Verbreitung von Plastik-Karten begann Anfang der 50er Jahre in den USA. Der preisgünstige Kunststoff PVC ermöglichte die Herstellung von robusten, langlebigen Karten, die für den Gebrauch im täglichen Leben weit besser geeignet waren als die bis dahin gebräuchlichen Karten aus Papier oder Karton, welche mechanischen Belastungen und Klimaeinwirkungen nur unzureichend standhielten.

Die erste Vollplastik-Karte für den überregionalen Zahlungsverkehr wurde 1950 vom Diners-Club ausgegeben. Sie war für einen exklusiven Personenkreis bestimmt, diente somit auch als Statussymbol und ermöglichte es dem Karteninhaber, statt mit Bargeld mit seinem „guten Namen" zu bezahlen. Die Akzeptanz dieser Karten war zunächst auf Restaurants und Hotels der gehobenen Klasse beschränkt, weshalb sich für diesen Kartentyp der Begriff „Traveller and Entertainment Card" eingebürgert hat.

Durch den Eintritt von VISA und Mastercard in die Kartenszene verbreitete sich das Plastikgeld sehr rasch, zunächst in Amerika, einige Jahre später auch in Europa und dem Rest der Welt.

Heute ermöglichen es die Plastikkarten dem Reisenden, weltweit ohne Bargeld einzukaufen. Der Karteninhaber ist jederzeit zahlungsfähig, ohne das Risiko des Geldverlustes durch Diebstahl oder andere, gerade auf Reisen schwer abschätzbare Gefahren eingehen zu müssen. Außerdem entfällt der lästige Geldumtausch bei Auslandsreisen. Diese einzigartigen Vorteile verhalfen den Plastikkarten zu einer schnellen Verbreitung weltweit. Heute werden jährlich viele hundert Millionen Karten produziert und ausgegeben.

Die Funktion der Karten war zu Beginn recht einfach. Sie dienten zunächst als gegen Fälschung und Manipulation geschützter Datenträger, wobei die allgemeinen Daten, wie z. B. der Name des Kartenausgebers, aufgedruckt waren, während individuelle Daten, wie z. B. der Name des Karteninhabers oder die Kartennummer, durch Hochprägung aufgebracht wurden. Darüber hinaus hatten viele Karten ein Unterschriftsfeld, auf welchem der Karteninhaber seine Referenzunterschrift leisten konnte. Der Fälschungsschutz wurde bei diesen Karten der ersten Generation durch visuelle Merkmale wie Sicherheitsdruck und Unterschriftsfeld hergestellt. Die Sicherheit des Systems hing infolgedessen ganz wesent-

lich von der Qualität und Sorgfalt des Personals in den Kartenakzeptanzstellen ab, was bei der anfänglichen Exklusivität der Karten jedoch kein großes Problem darstellte. Mit der zunehmenden Verbreitung der Karten reichte diese doch recht einfache Funktionalität nicht mehr aus, zumal auch die Gefährdung durch organisiertes Verbrechen stetig wuchs.

Zum einen erforderte der steigende Kostendruck im Handel und bei den Banken eine maschinenlesbare Karte, andererseits stiegen von Jahr zu Jahr die Verluste der Kartenausgeber durch Zahlungsunfähigkeit von Kunden oder durch Betrug. Es war somit offensichtlich, dass Betrugs- und Manipulationssicherheit sowie die Funktionalität erweitert und verbessert werden mussten.

Als erste Verbesserung wurde zunächst ein Magnetstreifen auf die Kartenrückseite aufgebracht, wodurch außer den visuellen Informationen noch zusätzliche digitalisierte Daten in maschinenlesbarer Form gespeichert werden konnten. Hierdurch wurde es möglich, die bis dahin erforderlichen Papierbelege auf ein Minimum zu reduzieren. Die Unterschrift des Kunden zur Personenidentifizierung ist bei den klassischen Kreditkartenanwendungen jedoch nach wie vor auf einem Papierbeleg erforderlich. Neue Anwendungen konnten jedoch so konzipiert werden, dass Papierbelege gänzlich überflüssig wurden, sodass das Ziel, die Zettelwirtschaft durch elektronische Datenverarbeitung zu ersetzen, endlich erreicht werden konnte. Die Benutzeridentifizierung, die bis dahin durch die Leistung der Unterschrift erfolgte, musste hierzu durch ein neues Verfahren ersetzt werden. Allgemein durchgesetzt hat sich die Verwendung einer persönlichen Geheimzahl (abgekürzt PIN für *personal identification number*), die mit einem Referenzwert verglichen wird. Der Leser kennt sicherlich dieses Verfahren von der Benutzung der Geldausgabeautomaten. Dieser Kartentyp (hochgeprägte Karte mit Magnetstreifen) hat auch heute noch die größte Verbreitung im Zahlungsverkehr.

Die Magnetstreifentechnik hat jedoch einen entscheidenden Nachteil: Die auf der Magnetpiste gespeicherten Daten können beliebig von jedem, der sich eine Schreib-/Lesevorrichtung für Magnetkarten verschafft, gelesen, gelöscht und geschrieben werden. Der Magnetstreifen eignet sich daher nicht zur Speicherung von geheimen Informationen. Zur Sicherstellung der Vertraulichkeit sowie der Manipulationssicherheit der Daten sind zusätzliche Techniken notwendig. Zum Beispiel kann der Referenzwert für die PIN nicht auf dem Magnetstreifen gespeichert werden, sondern muss entweder im Terminal oder im Host in einer gesicherten Umgebung aufbewahrt werden. Das ist der Grund dafür, dass die meisten Systeme, in denen Karten mit Magnetstreifen eingesetzt werden, aus Sicherheitsgründen eine Online-Verbindung zum Hostrechner des Systems haben, was jedoch hohe Kosten für die erforderliche Datenübertragung verursacht. Um die Kosten möglichst niedrig zu halten, wurde nach Lösungen gesucht, die es ermöglichen, Kartentransaktionen Offline durchführen zu können, ohne die Sicherheit des Systems zu gefährden.

Die Entwicklung der Chipkarte parallel zur Ausweitung der elektronischen Datenverarbeitung erschloss völlig neue Möglichkeiten zur Lösung dieses Problems.

Die rasanten Fortschritte der Mikroelektronik ermöglichten es, in den 70er Jahren Datenspeicher und Rechnerlogik auf einem einzigen kleinen Siliziumplättchen von wenigen Quadratmillimetern Fläche zu integrieren. Die Idee, solch einen integrierten Schaltkreis in eine Identifikationskarte einzubauen, wurde bereits 1968 von den Erfindern Jürgen Dethloff und Helmut Grötrupp in Deutschland zum Patent angemeldet. Im Jahre 1970 folgte eine ähnliche Anmeldung in Japan von Kunitaka Arimura. Richtig in Bewegung

kam die Entwicklung der Chipkarten jedoch erst, nachdem Roland Moreno 1974 seine Chipkartenpatente in Frankreich angemeldet hatte. Erst jetzt war die Halbleiterindustrie in der Lage, die erforderlichen integrierten Schaltungen zu akzeptablen Preisen zu liefern. Es gab aber immer noch eine Menge technologischer Probleme zu lösen, bis aus den ersten Prototypen mit teilweise mehreren integrierten Schaltkreisen ein marktgerechtes Produkt wurde, das zu angemessenen Preisen mit einer ausreichenden Zuverlässigkeit und Qualität in großer Stückzahl produziert werden konnte.

Da die grundlegenden Erfindungen für die Chipkartentechnik aus Deutschland und Frankreich stammen, ist es nicht verwunderlich, dass diese Länder die Vorreiterrolle in der Entwicklung und Vermarktung der Chipkarten spielten.

Der große Durchbruch wurde 1984 erzielt, als die französische PTT einen Feldversuch mit Telefonkarten erfolgreich durchführte. Bei diesem Feldversuch konnten die an die Chipkarte gestellten Erwartungen wie Manipulationssicherheit und hohe Zuverlässigkeit auf Anhieb unter Beweis gestellt werden. Bezeichnend war, dass der Durchbruch für die Chipkarte nicht in einem System geschafft wurde, in dem bereits konventionelle Karten zum Einsatz kamen, sondern in einer für Karten neuen Anwendung. Die Einführung einer neuen Technologie in einer neuen Anwendung hat den großen Vorteil, dass man nicht auf Kompatibilität zu vorhandenen Systemen Rücksicht nehmen muss und somit die neuen Möglichkeiten der neuen Technik uneingeschränkt nutzen kann.

In Deutschland fand 1984/85 ein Pilotversuch mit Telefonkarten verschiedener Technologien statt. Es kamen Karten mit Magnetstreifen, Karten mit optischer Speicherung (so genannte Hologrammkarten) und Chipkarten in einem vergleichenden Test zum Einsatz.

Aus diesem Pilotversuch ging die Chipkarte als Sieger hervor. Neben hoher Manipulationssicherheit und Zuverlässigkeit versprach die Chiptechnologie für die Zukunft die größte Flexibilität in der Anwendung. Während in den französischen Telefonkartenchips noch die ältere, aber preisgünstige EPROM-Technologie verwendet wurde, kamen in den deutschen Telefonkarten von Anfang an die neuen EEPROM-Chips zum Einsatz, die keine externe Programmierspannung mehr benötigen. Das hat aber leider dazu geführt, dass die französischen und deutschen Telefonkarten nicht kompatibel sind. Daher ist zu befürchten, dass auch nach der Einführung des Euro kurzfristig deutsche und französische Telefonkarten nicht im jeweils anderen Land benutzt werden können.

Nach diesen erfolgreichen Versuchen mit Telefonkarten in Frankreich und dann in Deutschland ging es mit atemberaubender Geschwindigkeit weiter. Im Jahr 1986 waren in Frankreich bereits mehrere Millionen Chipkarten zum Telefonieren im Umlauf. 1990 waren es fast 60 Millionen und 1997 weltweit mehrere hundert Millionen.

Deutschland erlebte eine ähnliche Entwicklung mit einer Verzögerung von ungefähr drei Jahren. Nach der erfolgreichen Einführung der öffentlichen Chipkartentelefone in Frankreich und Deutschland wurden diese Systeme weltweit vermarktet. Telefonkarten mit Chip kommen heute in über 50 Ländern weltweit zum Einsatz.

Bei den in den Telefonkarten eingesetzten elektronischen Schaltkreisen handelt es sich um relativ einfache, kleine und damit auch kostengünstige Speicherchips mit einer speziellen Sicherheitslogik, die das manipulationssichere Abbuchen von Werteinheiten ermöglicht. Aber auch die wesentlich komplexeren und größeren Mikroprozessorchips wurden zuerst in der Telekommunikation, und zwar im Mobilfunk in größerem Umfang eingesetzt. Hier spielte die Deutsche Bundespost im Jahr 1988 den Vorreiter mit der Einführung einer

modernen Mikroprozessorkarte in EEPROM-Technologie als Berechtigungskarte für das analoge Mobilfunknetz (C-Netz). Der Grund für die Einführung waren zunehmende Betrugsfälle mit den bis dahin eingesetzten Magnetstreifenkarten. Das analoge Mobilfunknetz war aus technischen Gründen auf relativ wenige Teilnehmer (etwa 1 Million) begrenzt, sodass dies noch kein wirklicher Massenmarkt für Mikroprozessorkarten war. Allerdings waren die positiven Erfahrungen, die im analogen Mobilfunk mit Chipkarten gewonnen werden konnten, entscheidend für die Einführung der Chipkarten im digitalen GSM-Netz, welches 1991 in mehreren Ländern Europas in Betrieb genommen wurde und heute weltweit mit mehr als 600 Millionen Teilnehmern in über 170 Ländern verbreitet ist.

Bei den Bankkarten verlief die Entwicklung wesentlich langsamer, was unter anderem auch auf die höhere Komplexität der Bankkarten im Vergleich zu den Telefonkarten zurückzuführen ist. Diese Unterschiede werden in den folgenden Kapiteln noch im Detail erläutert. An dieser Stelle sei nur darauf hingewiesen, dass für die Verbreitung der Bankkarte neben der Halbleitertechnologie die Entwicklung der modernen Kryptographie von entscheidender Bedeutung war.

Mit der allgemeinen Verbreitung der elektronischen Datenverarbeitung in den 60er Jahren erlebte die Entwicklung der Kryptographie sozusagen einen Quantensprung. Die moderne Hard- und Software ermöglichte die Implementierung von komplexen und anspruchsvollen mathematischen Algorithmen, mit denen ein Grad an Sicherheit erreicht wurde, wie er bis dahin ohne Beispiel war. Was noch hinzukam, war, dass diese neue Technik für jedermann verfügbar wurde, während bis dahin die Kryptographie eine Geheimwissenschaft für das Militär und die Geheimdienste war.

Durch den Einsatz dieser modernen kryptographischen Verfahren wurde die Stärke der Sicherheitsmechanismen in der elektronischen Datenverarbeitung berechenbar. Man war nicht mehr auf die doch stark subjektive Bewertung herkömmlicher Verfahren angewiesen, deren Sicherheit wesentlich auf der Geheimhaltung der angewandten Verfahren beruhte.

Die Chipkarte war das ideale Medium, welches die hohe Sicherheit auf der Basis der Kryptographie für jedermann zugänglich machen konnte, weil sie geheime Schlüssel vor unbefugtem Zugriff geschützt speichern und gleichzeitig Kryptoalgorithmen ausführen kann. Außerdem sind Chipkarten so klein und einfach in der Handhabung, dass sie von jedermann im täglichen Leben überall mitgeführt und benutzt werden können. Es war naheliegend, dass man versuchte, diese neuen sicherheitstechnischen Möglichkeiten im kartengestützten Zahlungsverkehr zu nutzen, um die mit zunehmender Verbreitung der Magnetstreifenkarten wachsenden Sicherheitsrisiken in den Griff zu bekommen.

Als erste entschieden sich die französischen Banken im Jahre 1984 dazu, diese neue faszinierende Technik einzuführen, nachdem bereits 1982/83 ein Pilotversuch mit 60 000 Karten stattgefunden hatte. Es dauerte dann noch 10 Jahre, bis alle französischen Bankkarten mit Chip ausgerüstet waren. In Deutschland fand 1984/85 ein erster Feldversuch mit einer multifunktionalen Zahlungsverkehrskarte mit Chip statt. Es sollte dann aber noch bis 1996 dauern, bis der Zentrale Kreditausschuss (ZKA) eine Spezifikation für die multifunktionale eurocheque-Karte mit Chip herausgab. Im Jahr 1997 wurden dann von allen Sparkassen und vielen Banken die neuen Chipkarten herausgegeben. Ein Jahr zuvor wurden bereits in Österreich multifunktionale Chipkarten mit POS-Funktion, elektronischer Geldbörse sowie möglichen Zusatzanwendungen landesweit herausgegeben, sodass Österreich weltweit das erste Land mit einem flächendeckenden elektronischem Geldbörsensystem wurde.

Ein wichtiger Meilenstein für die zukünftige weltweite Verbreitung der Chipkarten im Zahlungsverkehr war die Verabschiedung der so genannten EMV-Spezifikation, die gemeinsam von Europay, Mastercard und Visa erarbeitet wurde und in der ersten Version 1994 veröffentlicht wurde. Sie beschreibt im Detail Kreditkarten mit einem Mikroprozessorchip und gewährleistet die Kompatibilität der zukünftigen Chipkarten der drei großen Kreditkartenorganisationen.

Als weiteres Zugpferd für die internationale Verbreitung der Chipkarten im Zahlungsverkehr haben sich die elektronischen Geldbörsensysteme erwiesen. In Dänemark wurde 1992 das erste System mit dem Namen Danmønt in Betrieb genommen. Bis heute gibt es allein in Europa mehr als 20 nationale Systeme, von denen sich viele an der europäischen Norm EN 1546 orientieren. Aber auch außerhalb Europas nehmen die Anwendungen zu. Selbst in den USA, wo die Chipkarte bisher kaum Fuß fassen konnte, wurde bei den Olympischen Sommerspielen 1996 in Atlanta von Visa eine Chipkarte als elektronische Geldbörse erprobt. Neue vielversprechende Anwendungsmöglichkeiten für die elektronische Geldbörse bieten sich beim Bezahlen im Internet. Das Problem, im offenen Internet sicher, aber anonym weltweit auch kleine Beträge bezahlen zu können, ist bis heute noch nicht zufriedenstellend gelöst. Die Chipkarte kann bei der Lösung dieses Problems eine entscheidende Rolle spielen. Außerdem kann die Chipkarte eine wichtige Funktion bei der Einführung der elektronischen Unterschrift übernehmen, mit der in verschiedenen europäischen Ländern begonnen wurde, nachdem im Jahre 1999 durch die Verabschiedung der EU-Direktive für elektronische Signaturen durch das europäische Parlament die rechtliche Grundlage zur Verwendung digitaler Signaturen geschaffen wurde.

Eine andere Anwendung hat in Deutschland dafür gesorgt, dass heute beinahe jeder eine Chipkarte besitzt. Bei der Einführung der Krankenversichertenkarten mit Chip wurden über 70 Millionen Chipkarten an alle gesetzlich Versicherten ausgegeben. Mittlerweile sind weltweit in vielen Ländern Chipkarten im Gesundheitsbereich im Einsatz.

Die hohe funktionale Flexibilität der Chipkarte, die so weit geht, dass eine bereits in Benutzung befindliche Karte für neue Anwendungen nachprogrammiert werden kann, hat über die traditionellen Anwendungen von Karten hinaus völlig neue Einsatzbereiche erschlossen.

Auch im öffentlichen Personenverkehr haben sich die Chipkarten als elektronischer Fahrschein in vielen Städten der Erde etabliert. Hierzu werden in der Regel kontaktlose Chipkarten eingesetzt, da sie in der Handhabung besonders einfach und kundenfreundlich sind.

1.2 Anwendungsgebiete

Wie bereits in dem geschichtlichen Überblick dargestellt wurde, sind die Anwendungsmöglichkeiten von Chipkarten äußerst vielseitig und nehmen mit der steigenden Rechenleistung und Speicherkapazität der verfügbaren integrierten Schaltungen noch stetig zu. Es ist unmöglich, im Rahmen dieses Buches alle Anwendungen im Detail zu erläutern. Stattdessen sollen an einigen typischen Beispielen die wesentlichen Eigenschaften der Chipkarten veranschaulicht werden. In diesem einleitenden Kapitel soll nur ein erster Überblick über die vielfältige Verwendbarkeit gegeben werden. Einige beispielhafte Anwendungen werden in den Kapiteln 12, 13 und 14 detaillierter dargestellt.

Um sich einen besseren Überblick zu verschaffen, ist die Einteilung der Chipkarten in Speicherkarten und Mikroprozessorkarten hilfreich.

1.2.1 Speicherkarten

Die ersten Chipkarten, die in großer Menge zum Einsatz kamen, waren Speicherkarten für die Telefonanwendung. Diese Karten werden im Voraus bezahlt, und der im Chip elektronisch gespeicherte Wert wird bei der Benutzung um den jeweils verbrauchten Betrag reduziert. Es muss natürlich verhindert werden, dass der Benutzer einer solchen Karte den gespeicherten Wert wieder erhöhen kann, was bei einer Magnetstreifenkarte problemlos möglich wäre. Der Benutzer müsste sich nur die beim Kauf der Karte gespeicherten Daten merken, um diese nach Benutzung der Karte wieder auf den Magnetstreifen zu schreiben. Die Karte hätte dann wieder den ursprünglichen Wert und könnte erneut benutzt werden. Dieser auch „Buffern" genannte Angriff wird bei Chipkarten durch eine Sicherheitslogik im Chip verhindert, die es unmöglich macht, dass eine einmal geschriebene Speicherzelle wieder gelöscht werden kann. Die Reduzierung des Wertes der Karte um die verbrauchten Einheiten ist somit irreversibel.

Chipkarten dieses Typs lassen sich außer zum Telefonieren natürlich überall einsetzen, wo gegen Vorbezahlung eine Ware oder Dienstleistung bargeldlos verkauft werden soll. Beispiele hierfür sind der öffentliche Personennahverkehr, Verkaufsautomaten aller Art, Kantinen, Schwimmbäder, Parkgebühren und vieles andere. Der Vorteil dieses Kartentyps liegt in der einfachen Technik (die Chipfläche beträgt typischerweise nur wenige Quadratmillimeter) und dem damit verbundenen günstigen Preis. Der Nachteil liegt darin, dass die Karte nach Verbrauch des Wertes nicht mehr verwendet werden kann, sondern als Müll entsorgt werden muss, soweit sie nicht in einer Kartensammlung landet.

Eine weitere typische Anwendung von Speicherkarten ist die Krankenversichertenkarte, die seit 1994 an alle Versicherten in Deutschland ausgegeben wird. Bei dieser Karte sind im Chip die Daten gespeichert, die bisher auf dem Krankenschein eingetragen waren und die jetzt auch auf der Karte aufgedruckt bzw. mittels eines Laserstrahls geschrieben sind. Die Speicherung im Chip macht die Karte mit einfachen Mitteln maschinenlesbar.

Zusammenfassend halten wir fest: Speicherchipkarten sind von der Funktion beschränkt. Sie ermöglichen es, durch die integrierte Sicherheitslogik die gespeicherten Daten vor Manipulation zu schützen. Sie eignen sich als Wertkarten oder Ausweiskarten in Systemen, bei denen es besonders auf einen günstigen Kartenpreis ankommt.

1.2.2 Mikroprozessorkarten

Die erste Anwendung von Mikroprozessorkarten war – wie bereits erwähnt – die Anwendung als Bankkarte in Frankreich. Die Möglichkeit, geheime Schlüssel sicher speichern zu können und moderne Kryptoalgorithmen ausführen zu können, ermöglichte die Realisierung von Offline-Zahlungssystemen mit hohem Sicherheitsniveau.

Da der Mikroprozessor in der Karte frei programmiert werden kann, ist die Funktionalität von Prozessorkarten nur durch den verfügbaren Speicherplatz und die Rechenleistung des Rechenwerkes beschränkt. Der Phantasie sind bei der Implementierung von Chipkartensystemen somit nur diese technologischen Grenzen gesetzt, die sich mit jeder neuen Generation von integrierten Schaltungen stark erweitern.

Nachdem in Folge der Massenproduktion die Preise für Mikroprozessorkarten bis Anfang der 90er Jahre stark gefallen waren, wurden auch tatsächlich jedes Jahr neue Anwen-

dungen erschlossen. Eine besondere Bedeutung für die internationale Verbreitung der Chipkarten kam hierbei der Anwendung im Mobiltelefon zu.

Nachdem die Chipkarte bereits im nationalen deutschen C-Netz (analoges Mobiltelefonnetz) die Bewährungsprobe für den Einsatz in mobilen Endgeräten bestanden hatte, wurde sie als Medium für den Zugang zum digitalen europäischen Mobiltelefon (GSM) vorgeschrieben. Mit der Chipkarte war es möglich, zum einen ein Höchstmaß an Sicherheit für den Zugang zum Mobiltelefonnetz zu erreichen. Andererseits bot die Chipkarte neue Möglichkeiten und damit große Vorteile bei der Vermarktung des Mobiltelefons, weil sie die einfache Trennung des Verkaufs von Geräten und des Verkaufs der Dienstleistungen durch die Netzbetreiber und Diensteanbieter ermöglichte. Ohne die Chipkarte hätte sich das Mobiltelefon in Europa sicher nicht so schnell verbreiten und zum Weltstandard entwickeln können.

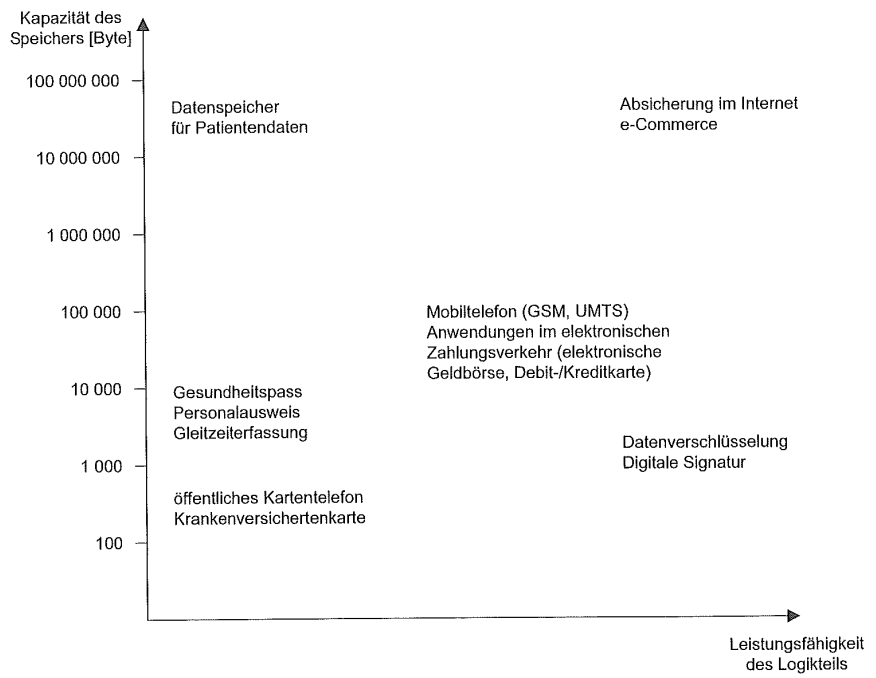


Bild 1.1 Typische Anwendungsfelder von Chipkarten und dazu erforderliche Speicherkapazität und Rechenleistung.

Weitere beispielhafte Anwendungen für Mikroprozessorkarten sind Ausweiskarten, Zugangskontrolle, Zugriffskontrolle auf Rechner, geschützte Datenspeicher, elektronische Unterschrift, elektronische Geldbörse sowie multifunktionale Karten, die mehrere Anwendungen in einer Karte enthalten. Moderne Chipkarten-Betriebssysteme ermöglichen es, auch nach der Ausgabe der Chipkarte an den Benutzer noch neue Anwendungen auf die Karte laden zu können, ohne die Sicherheit der verschiedenen Anwendungen zu gefährden. Mit dieser neuen Flexibilität erschließen sich völlig neue Anwendungsfelder. Um z. B. den

Handel und das Bezahlen im Internet auf eine vertrauenswürdige Basis zu stellen, werden unbedingt persönliche Sicherheitsmodule benötigt, in denen die persönlichen Schlüssel geschützt aufbewahrt werden können und die leistungsfähige Verschlüsselungsalgorithmen rechnen können. Diese Aufgabe kann eine Mikroprozessorkarte mit einem Kryptoprozessor elegant lösen. Zurzeit wird weltweit an Spezifikationen für sichere Anwendungen mit Chipkarten im Internet gearbeitet. Es ist zu erwarten, dass in wenigen Jahren jeder PC mit einem Chipkarteninterface ausgestattet ist.

Zusammenfassend halten wir fest: Die wesentlichen Vorzüge der Mikroprozessorkarte sind hohe Speicherkapazität, die sichere Speicherung geheimer Daten und die Fähigkeit, Kryptoalgorithmen rechnen zu können. Diese Vorzüge ermöglichen außer der traditionellen Anwendung als Bankkarte eine Fülle neuer Anwendungen. Das Potenzial der Chipkarten ist heute noch längst nicht ausgeschöpft und wird obendrein durch den Fortschritt der Halbleitertechnologie ständig erweitert.

1.2.3 Kontaktlose Karten

In den letzten Jahren wurden die kontaktlosen Karten, bei denen die Energie- und Datenübertragung ohne galvanische Kopplung zwischen Karte und Terminal erfolgt, zur Produktionsreife entwickelt, sodass heute sowohl Speicherkarten als auch Mikroprozessorkarten verfügbar sind. Während die kontaktlosen Mikroprozessorkarten wegen der höheren Leistungsaufnahme meist nur über eine Entfernung von wenigen Zentimetern funktionieren, ermöglichen kontaktlose Speicherkarten einen Betrieb bis zu einem Meter Abstand vom Terminal. Das bedeutet, dass die Karte zum Betrieb nicht unbedingt in die Hand genommen werden muss, sondern z. B. in der Geldbörse oder der Brieftasche bleiben kann. Deshalb eignen sich diese Karten besonders für Anwendungen, bei denen Personen oder Gegenstände schnell identifiziert werden sollen. Beispielhafte Anwendungen sind:

- Zugangskontrolle
- öffentlicher Personennahverkehr
- Skipass
- Flugticket
- Gepäckidentifikation

Es gibt jedoch auch Anwendungen, bei denen die Funktion über eine größere Entfernung kritisch ist und deshalb verhindert werden muss. Ein typisches Beispiel hierfür ist die elektronische Geldbörse. Um den Bezahlvorgang durchzuführen, ist in der Regel eine Willenserklärung des Karteninhabers erforderlich, in der der Betrag und die Einwilligung zur Zahlung bestätigt wird. Bei der kontaktbehafteten Karte stellt das Einführen der Karte in das Terminal und die Bestätigung des angezeigten Betrages auf dessen Tastatur diese Willenserklärung dar. Wäre ein Bezahlvorgang kontaktlos über eine größere Entfernung möglich, könnte ein Betrüger vollkommen unbemerkt vom Karteninhaber Geld aus der Karte abbuchen. Eine mögliche Lösung für dieses Problem bieten die so genannten Dual-Interface-Karten (auch Combicard genannt). In ihnen werden beide Funktionen, die der kontaktbehafteten und die der kontaktlosen Chipkarten, auf einem einzigen Chip vereint. Eine solche Karte kann somit wahlweise über die kontaktbehaftete oder die kontaktlose Schnittstelle mit dem Terminal kommunizieren.

Großes Interesse an solchen Karten besteht beim öffentlichen Personennahverkehr. Erweitert man die Chipkarten des Zahlungsverkehrs, die in der Regel kontaktbehaftete Karten sind, um die Funktion des elektronischen Fahrscheins über eine kontaktlose Schnittstelle, so können die Verkehrsbetriebe auf die vom Kreditgewerbe ausgegebenen Karten und deren Infrastruktur zurückgreifen.

1.3 Normung

Die Voraussetzung für die weltweite Verbreitung der Chipkarte im täglichen Leben, wie wir sie heute in Deutschland in Form der Telefonkarten, Krankenversicherungskarten und Zahlungsverkehrskarten haben, war die Schaffung von nationalen und internationalen Normen. Wegen der besonderen Bedeutung der Normen werden wir in diesem Buch immer wieder auf die gültigen, aber auch auf die in Arbeit befindlichen Normen verweisen. Warum sind Normen so wichtig für die Verbreitung der Chipkarte?

Die Chipkarte spielt in der Regel die Rolle einer Systemkomponente in einem komplexen System. Das bedeutet, dass die Schnittstellen zwischen der Karte und dem System genau spezifiziert und aufeinander abgestimmt sein müssen. Natürlich ist es möglich, dies für jedes System im Einzelfall zu tun, ohne auf andere Systeme Rücksicht zu nehmen. Das würde aber bedeuten, dass für jedes System eine spezielle Chipkarte benötigt würde. Für jedes System müsste eine spezielle Karte entwickelt werden, und der Anwender müsste für jede Anwendung eine spezielle Karte bei sich haben. Um dies zu vermeiden, wurde versucht, anwendungsunabhängige Normen zu entwickeln, die eine multifunktionale Chipkarte möglich machen. Da die Chipkarte meist diejenige Komponente des Systems darstellt, die der Benutzer in der Hand hält, kommt ihr eine hervorragende Bedeutung für die Bekanntheit und die Akzeptanz des gesamten Systems zu. Aus technischer und organisatorischer Sicht ist die Chipkarte jedoch meist nur die Spitze des Eisberges. Hinter dem Kartenendgerät verbergen sich nämlich häufig komplexe, meist vernetzte Systeme, die den Kundennutzen überhaupt erst ermöglichen.

Nehmen wir als Beispiel die aus technischer Sicht recht einfache Telefonkarte: Für sich allein genommen ist sie fast wertlos, allenfalls als Sammlerobjekt zu gebrauchen. Ihren Nutzen – nämlich ohne Münzen an öffentlichen Fernsprechern telefonieren zu können – kann sie erst entfalten, wenn zigtausende von Endgeräten flächendeckend installiert und an ein Netz angeschlossen sind. Die hohen Investitionen, die hierzu erforderlich sind, lassen sich erst dann rechtfertigen, wenn entsprechende Normen und Spezifikationen die Zukunftssicherheit eines solchen Systems gewährleisten. Unabdingbare Voraussetzungen sind Normen für multifunktionale Chipkarten, die für verschiedene Anwendungen, wie z. B. Telefonieren, elektronische Geldbörse, elektronischer Fahrschein etc., benutzt werden können.

Was sind Normen?

Diese Frage ist nicht so trivial, wie es auf den ersten Blick erscheinen mag, zumal da die Begriffe „Normung", „Standard" und „Spezifikation" im Deutschen recht bedenkenlos durcheinandergeworfen werden, was noch dadurch verstärkt wird, dass der englische Begriff „standard" meist mit „Norm" und nicht mit „Standard" übersetzt werden muss. Schauen wir uns zur Klärung die Definition bei ISO/IEC an:

Norm: Dokument, das mit Konsens erstellt und von einer anerkannten Institution angenommen wurde und das für die allgemeine und wiederkehrende Anwendung Regeln, Leitlinien oder Merkmale für Tätigkeiten oder deren Ergebnisse festlegt, wobei ein optimaler Ordnungsgrad in einem gegebenen Zusammenhang angestrebt wird.

Anmerkung: Normen sollten auf den gesicherten Ergebnissen von Wissenschaft, Technik und Erfahrung basieren und auf die Förderung optimaler Vorteile für die Gesellschaft abzielen.

Internationale Normen sollen dazu beitragen, das Leben einfacher zu machen und die Zuverlässigkeit und Nützlichkeit der Produkte und Dienstleistungen zu steigern.

Und um Unklarheiten von vornherein zu vermeiden, ist auch der Begriff „Konsens" bei ISO/IEC definiert:

Konsens: Allgemeine Zustimmung, die durch das Fehlen aufrechterhaltenen Widerspruches gegen wesentliche Inhalte seitens irgendeines wichtigen Anteils der betroffenen Interessen und durch ein Verfahren gekennzeichnet ist, das versucht, die Gesichtspunkte aller betroffenen Parteien zu berücksichtigen und alle Gegenargumente auszuräumen.

Anmerkung: Konsens bedeutet nicht notwendigerweise Einstimmigkeit.

Obwohl hier keine Einstimmigkeit gefordert wird, braucht der demokratische Konsensweg natürlich etwas Zeit, zumal nicht nur die Techniker, sondern alle betroffenen Parteien gehört werden müssen, da Normen ja die Förderung optimaler Vorteile für die Gesellschaft zum Ziel haben. Die Fertigstellung einer ISO- oder CEN-Norm dauert deshalb meist mehrere Jahre. Die Trägheit dieses Prozesses hat häufig zur Folge, dass ein eingeschränkter Interessentenkreis, z. B. die Industrie, eigene Spezifikationen erstellt, so genannte Industriestandards, um eine schnellere Systemeinführung zu ermöglichen. Dies gilt besonders für den Bereich der Informationstechnologie, der sich durch besonders schnelle Entwicklung und damit besonders kurze Innovationszyklen auszeichnet. Industriestandards haben zwar den Vorteil, dass sie wesentlich schneller verfügbar sind als Normen, bergen aber die Gefahr in sich, dass wichtige Interessen derer, die nicht am Entstehungsprozess beteiligt sind, unberücksichtigt bleiben. Bei der ISO versucht man deshalb Möglichkeiten zu schaffen, dass wichtige öffentlich zugängliche Spezifikationen nachträglich in die Normung eingebracht werden können.

Was ist ISO/IEC?

Für die Chipkarte sind die relevanten ISO/IEC-Normen von besonderer Bedeutung, da sie die grundlegenden Eigenschaften der Chipkarten festlegen. Doch was versteckt sich hinter den Abkürzungen ISO/IEC?

ISO steht für International Organization of Standardisation und IEC für International Electrotechnical Commission.

Die internationale Organisation für Standardisierung (ISO) ist ein weltweiter Verband von ca. 100 nationalen Standardisierungsgremien, jeweils einem pro Land. Die ISO wurde 1947 gegründet und ist eine nichtstaatliche Organisation. Sie hat den Auftrag, die Entwicklung von Normen weltweit voranzutreiben mit dem Ziel, den internationalen Austausch

von Gütern und Dienstleistungen zu erleichtern und die Zusammenarbeit im wissenschaftlichen, technologischen und wirtschaftlichen Bereich zu entwickeln.

Die Ergebnisse der Arbeit von ISO sind internationale Übereinkünfte, die als ISO-Normen veröffentlicht werden.

Der Name ISO ist übrigens kein Akronym. Ein Akronym für den offiziellen Titel „International Organization of Standardization" müsste ja IOS heißen. Vielmehr wurde „ISO" aus dem griechischen „isos" abgeleitet, was „gleich" bedeutet. Die daraus abgeleitete Vorsilbe „iso-" ist in den drei offiziellen Sprachen der ISO – Englisch, Französisch und Russisch – sowie in vielen anderen Sprachen gebräuchlich.

Wie bereits erwähnt, sind die Mitglieder der ISO die nationalen Standardisierungsgremien der einzelnen Länder, wobei nur ein Gremium pro Land zugelassen ist. Die Mitgliedsorganisationen haben vier grundsätzliche Aufgaben:

- Information möglicher Interessenten im jeweiligen Land über relevante Aktivitäten und Möglichkeiten der internationalen Normung;
- Bildung einer abgestimmten nationalen Meinung und deren Vertretung bei den internationalen Verhandlungen;
- die Einrichtung eines Sekretariates für diejenigen ISO-Komitees, an denen das Land ein besonderes Interesse hat;
- Zahlung des Landesbeitrags zur Finanzierung der zentralen ISO-Organisation.

Deutschland wird in der ISO durch den DIN vertreten.

IEC ist ein Normungsorganisation, die den Bereich Elektrotechnik und Elektronik abdeckt. Die ersten Kartennormen wurden von der ISO herausgegeben. Nachdem die Chipkarte eingeführt wurde, gab es thematisch eine Überschneidung zwischen ISO und IEC. Um Doppelarbeit zu vermeiden, werden die Normen in einem gemeinsamen technischen Komitee erarbeitet und als ISO/IEC-Normen veröffentlicht.

Wie entsteht eine ISO-Norm?

Der Bedarf für eine Norm wird in der Regel durch einen Industriebereich bei der nationalen Normungsorganisation angemeldet, die diesen bei der ISO als neues Arbeitsthema vorschlägt. Wird der Vorschlag von der zuständigen Arbeitsgruppe, die aus technischen Experten der an dem Thema interessierten Länder besteht, angenommen, so wird zunächst das Ziel der zukünftigen Norm definiert.

Nachdem Übereinstimmung erzielt wurde, welche technischen Aspekte in der Norm berücksichtigt werden sollen, werden in der zweiten Phase die Detailspezifikationen der Norm zwischen den Ländern diskutiert und verhandelt. Ziel ist es, dabei möglichst einen Konsens aller beteiligten Länder zu erreichen. Das Ergebnis dieser Phase ist ein „Draft International Standard" (DIS).

Die letzte Phase beinhaltet die formale Abstimmung über den Normentwurf. Für die Annahme ist die Zustimmung von zwei Drittel der ISO Mitglieder, die aktiv an der Norm mitgearbeitet haben, erforderlich, sowie von 75 % aller Mitglieder, die an der Abstimmung teilnehmen. Bei Annahme wird der abgestimmte Text als ISO-Norm veröffentlicht.

Um einer Veralterung der ISO-Normen durch technische Weiterentwicklung vorzubeugen, sehen die ISO-Regeln spätestens nach fünf Jahren eine Überprüfung und gegebenenfalls Überarbeitung der Normen vor.

Zusammenarbeit mit IEC und CEN

Die ISO ist nicht die einzige internationale Normungsorganisation. Um Doppelarbeit zu vermeiden, arbeitet sie sehr eng mit der „International Electrotechnical Commission", IEC, zusammen. Die Zuständigkeitsbereiche wurden folgendermaßen definiert: IEC deckt den Bereich Elektrotechnik und Elektronik ab, für alle anderen Bereiche ist die ISO zuständig. Zu Themen von gemeinsamem Interesse werden vereinte technische Arbeitsgruppen gebildet, die gemeinsame ISO/IEC-Normen erarbeiten. Hierzu zählen auch die meisten Chipkartennormen.

Auch zwischen ISO und CEN (*Comité Européen de Normalisation*), dem europäischen Komitee zur Standardisierung, wurden Regeln für die Entwicklung von Normen vereinbart, die sowohl als europäische wie auch als internationale Normen anerkannt werden. Hierdurch können Zeit und Kosten eingespart werden.

Die internationale Chipkartennormung

Die internationale Chipkartennormung findet unter dem Dach von ISO/IEC und auf europäischer Ebene von CEN statt. Die wichtigsten Industrieländer sind in allen relevanten Gremien vertreten und unterhalten in der Regel Spiegelgremien als nationale Arbeits- und Abstimmungsgremien. Für Deutschland ist der DIN zuständig. Die Abbildung 1.2 gibt einen Überblick über die Struktur der relevanten ISO-Arbeitsgruppen und die von ihnen betreuten Normen.

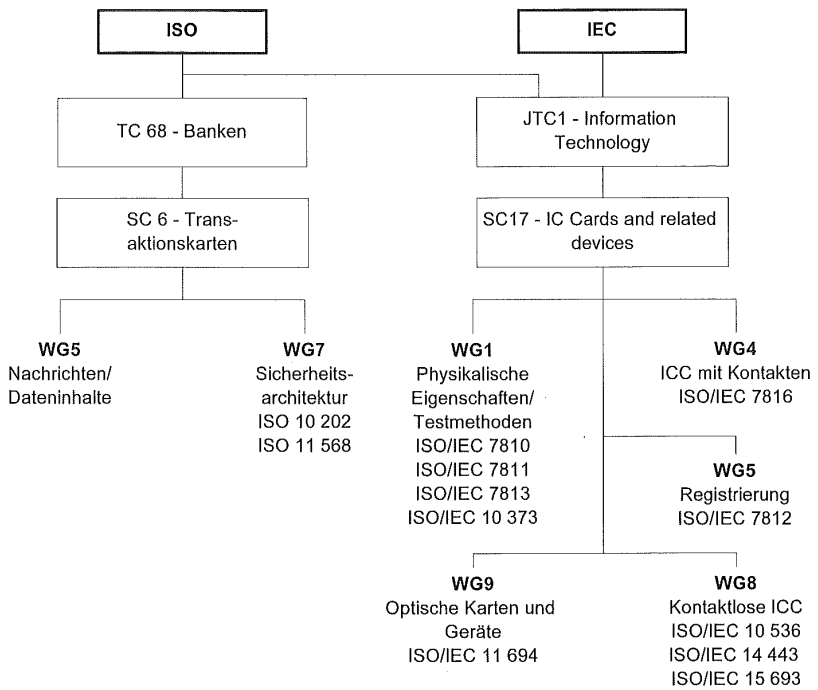


Bild 1.2 Überblick und Hierarchie der Arbeitsgruppen in der weltweiten Chipkartennormung.

Wie man sieht, gibt es zwei technische Komitees, die sich mit der Chipkartennormung befassen: Zum einen das ISO TC68/SC6, welches für die Normung der Transaktionskarten im Zahlungsverkehr zuständig ist, zum anderen ISO/IEC JTC1/SC17, welches für allgemeine Anwendungen verantwortlich ist. Diese Zweiteilung hat historische Gründe, da die ersten internationalen Anwendungen für Identifikationskarten im Bereich des Zahlungsverkehrs waren. Mittlerweile haben sich die Anwendungen jedoch stark ausgeweitet, sodass heute die allgemeinen Normen, die vom SC17 betreut werden, die größere Bedeutung haben und die bankenspezifischen Normen im Wesentlichen als eine Untermenge der allgemeinen Normen betrachtet werden können. Eine Kurzbeschreibung und der aktuelle Status der in Bild 1.2 aufgeführten Normen findet sich im Anhang.

Beim CEN wird das Thema Chipkarten im TC 224 (Maschinenlesbare Karten und zugehörige Geräteschnittstellen und Verfahren) bearbeitet.

Die Arbeit beim CEN ist eine Ergänzung der Arbeit bei ISO. Die ISO-Normen werden nach Möglichkeit als CEN-Normen übernommen, hierzu müssen sie z. B. in die drei offiziellen CEN-Sprachen (Deutsch, Englisch, Französisch) übersetzt werden. Bei Bedarf werden sie jedoch auch um europaspezifische Teile ergänzt oder auch eingeschränkt. Darüber hinaus werden von den CEN-Arbeitsgruppen anwendungsorientierte Normen erstellt, die bei der ISO in dieser Form nicht möglich wären.

Eine weitere europäische Normungsinstanz, das „European Telecommunications Standards Institute“ mit der Abkürzung ETSI, hat wesentlich zur internationalen Verbreitung der Chipkarten beigetragen. ETSI ist die Normungsinstanz der europäischen Telekommunikationsgesellschaften und der Telekommunikationsindustrie. Die Normreihe GSM 11.11 ff spezifiziert die Schnittstelle zwischen der Chipkarte, die im GSM-System mit Subscriber Identity Module (SIM) bezeichnet wird, und dem Mobiltelefon. Diese Normreihe baut auf den ISO/IEC Normen auf. Durch die weltweite Verbreitung des GSM Systems über Europa hinaus haben die ETSI Normen eine große Bedeutung für die Chipkartenindustrie.

Nach fast zwanzig Jahren Normungsarbeit sind heute die wichtigsten grundlegenden ISO-Normen für Chipkarten fertig und bilden die Basis für weitere, eher anwendungsorientierte Normen, die zur Zeit bei der ISO und CEN in Arbeit sind.

Grundlage waren die bereits existierenden ISO-Standards der Reihen 7810, 7811, 7812 und 7813, in denen die physikalischen Eigenschaften von Identifikationskarten im so genannten ID-1 Format definiert sind. Diese Standards schlossen hochgeprägte Karten und Karten mit Magnetstreifen ein, wie wir sie heute von den Kreditkarten her kennen.

Bei der Standardisierung der Chipkarten (in der ISO-Norm als integrated circuit(s) card (ICC) bezeichnet) wurde von Anfang an auf Kompatibilität zu diesen bestehenden Standards geachtet, wodurch ein problemloser Übergang von der Anwendung hochgeprägter Karten und Karten mit Magnetstreifen auf Chipkarten ermöglicht wird. Das ist möglich, weil alle Funktionselemente wie Hochprägung, Magnetstreifen, Kontakte und Interface-Elemente für kontaktlose Übertragung gleichzeitig in einer Karte integriert sein können. Dies hat allerdings auch zur Folge, dass empfindliche elektronische Bauelemente, wie sie integrierte Schaltungen nun einmal sind, harten Belastungen bei der Hochprägung von Karten oder den regelmäßigen Schlägen beim Abdruck der Hochprägung ausgesetzt sind. Dies stellt hohe Anforderungen an die Verpackung der integrierten Schaltkreise und deren Einbettung in die Karte.

Eine Übersicht der heute verfügbaren Normen mit einer kurzen Inhaltsangabe befindet sich im Anhang.[1]

In den letzten Jahren wurden vermehrt Spezifikationen von der Industrie oder anderen nicht öffentlichen Organisationen erarbeitet und veröffentlicht, ohne zu versuchen, sie in die Normungsarbeit der ISO einzubringen. Als Argument für diese Vorgehensweise wird meist angeführt, dass die Arbeitsweise der ISO zu langsam ist, um mit den kurzen Innovationszyklen der Informations- und Kommunikationsindustrie Schritt halten zu können. Da bei der Erstellung dieser Industriestandards häufig nur wenige Firmen beteiligt sind, besteht die große Gefahr, dass bei dieser Vorgehensweise die Interessen kleinerer Firmen und vor allem Interessen der Allgemeinheit nicht berücksichtigt werden können. Es ist eine große Herausforderung für die Zukunft der ISO, eine Arbeitsweise zu definieren, die es ermöglicht, allgemeine Interessen weiterhin zu wahren und trotzdem die Innovationsgeschwindigkeit nicht zu bremsen.

[1] Siehe auch Abschnitt 15.4 „Kommentiertes Normenverzeichnis".

2 Arten von Karten

Wie bereits in der Einleitung erwähnt, sind die Chipkarten das jüngste Kind der Familie der Identifikationskarten im ID-1 Format, wie sie in der Norm ISO 7810 „Identification Cards – Physical Characteristics" definiert sind. Diese Norm spezifiziert die physikalischen Eigenschaften von Identifikationskarten einschließlich der Materialeigenschaften wie Flexibilität, Temperaturbeständigkeit und Abmessungen für drei verschiedene Größen von Karten (ID-1, ID-2 und ID-3). Die Basis für die Chipkartennormen ISO 7816-1 ff bildet die ID-1 Karte, wie sie heute millionenfach als Karte für den Zahlungsverkehr verbreitet ist.

Dieses Kapitel gibt einen Überblick über verschiedene Arten von Karten im ID-1 Format. In vielen Anwendungen ist nämlich eine Kombination der Kartenfunktionen von besonderem Interesse, und zwar immer dann, wenn in einem bereits bestehenden System die vorhandenen Karten, wie z. B. Magnetstreifenkarten, durch Chipkarten abgelöst werden sollen. In der Regel ist es nämlich unmöglich, die vorhandene Infrastruktur – in diesem Falle die Terminals für Magnetstreifenkarten – von heute auf morgen durch eine neue Technik zu ersetzen.

Die Lösung wird im Allgemeinen darin bestehen, dass man für eine Übergangszeit Karten ausgibt, die mit Magnetstreifen und mit Chip ausgerüstet sind, sodass die Karten sowohl an den alten Magnetkartenterminals wie auch an den neuen Chipkartenterminals benutzt werden können. Neue Funktionen, die nur mit dem Chip möglich sind, können dann natürlich an den Magnetkartenterminals nicht genutzt werden.

2.1 Hochgeprägte Karten

Die Hochprägung ist die älteste Technik zur Beschriftung von Identifikationskarten in maschinenlesbarer Form. Die hochgeprägten Zeichen auf der Karte können mit einem einfachen und billigen Gerät durch Abdruck auf Papier übertragen werden. Auch das visuelle Lesen der Hochprägung ist ohne weiteres möglich. Die Art und Lage der Hochprägung ist im Standard ISO 7811 „Identification Cards – Recording Technics" spezifiziert. Dieser Standard besteht aus fünf Teilen und behandelt außer der Hochprägung auch den Magnetstreifen.

In ISO 7811, Teil 1 sind die Anforderungen an hochgeprägte Zeichen, wie deren Form, Größe und Prägehöhe, spezifiziert.

Im Teil 3 wird die genaue Lage der Zeichen auf der Karte festgelegt, und zwar sind zwei verschiedene Bereiche definiert. Der Bereich 1 ist für die Kartenidentifikationsnummer re-

serviert, durch welche der Kartenausgeber sowie der Karteninhaber festgelegt sind. Der Bereich 2 ist für weitere Daten des Karteninhabers wie z. B. Name und Adresse vorgesehen.

Auf den ersten Blick mag die Informationsübertragung durch Abdruck hochgeprägter Zeichen recht primitiv erscheinen. Die Einfachheit dieser Technik hat jedoch ihre weltweite Verbreitung auch in unterentwickelten Ländern der Erde ermöglicht. Die Anwendung dieser Technik erfordert weder elektrische Energie noch Anschluss an das Telefonnetz.

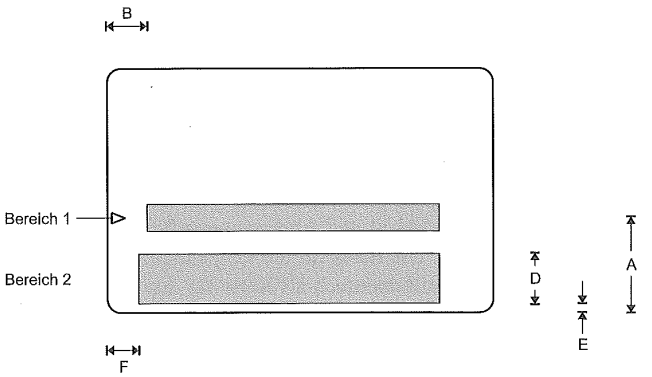


Bild 2.1 Lage der Hochprägung nach ISO 7811-3. Der Bereich 1 ist für die Identifikationsnummer (19 Zeichen) vorgesehen. Der Bereich 2 für Name und Adresse (4 · 27 Zeichen). Die folgenden Abmessungen sind in der Norm festgelegt:

A: 21,42 mm ± 0,12 mm B: 10,18 mm ± 0,25 mm
D: 14,53 mm E: 2,41 mm ... 3,30 mm
F: 7,65 mm ± 0,25 mm

2.2 Magnetstreifenkarten

Der wesentliche Nachteil der hochgeprägten Karten liegt darin, dass bei ihrer Nutzung eine Flut von Papierbelegen entsteht, deren Handhabung und Auswertung hohe Kosten verursachen. Abhilfe schafft hier die digitale Kodierung der Kartendaten auf einem Magnetstreifen, der sich auf der Rückseite der Karte befindet.

Zum Lesen des Magnetstreifens wird dieser von Hand oder maschinell an einem Lesekopf vorbeigezogen, wobei die Daten gelesen und elektronisch gespeichert werden. Zur Bearbeitung dieser Daten ist dann kein Papier mehr erforderlich.

In den Teilen 2, 4 und 5 der ISO-Norm 7811 sind die Eigenschaften des Magnetstreifens, die Kodiertechnik sowie die Lage der Magnetspuren spezifiziert. Insgesamt können sich drei Spuren auf dem Magnetstreifen befinden.

Die Spuren 1 und 2 sind nur für den Lesebetrieb spezifiziert, während die Spur 3 auch beschrieben werden kann.

Die Speicherkapazität des Magnetstreifens ist allerdings mit ca. 1000 Bit nicht sehr groß. Sie reicht jedoch leicht aus, um die Informationen der Hochprägung zu speichern. Auf der Spur 3 können zusätzliche Daten gelesen und geschrieben werden, wie z. B. Daten der letzten Transaktion bei Kreditkarten.

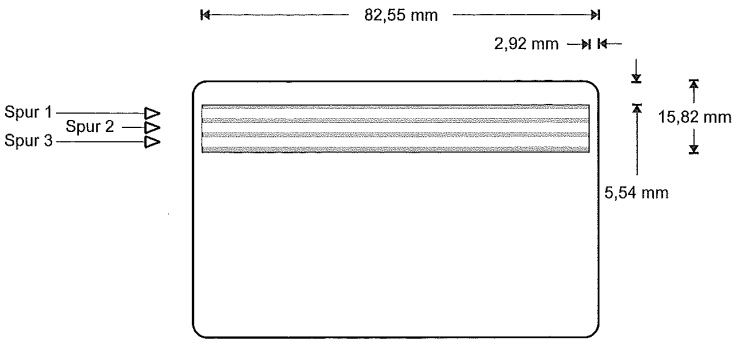


Bild 2.2 Die Lage des Magnetstreifens auf der ID-1 Karte. Der Datenbereich des Magnetstreifens reicht bewusst nicht bis zu den Kartenkanten, da die handbedienten Magnetstreifenleser dort zu einer sehr frühzeitigen Abnutzung des Magnetstreifens führen.

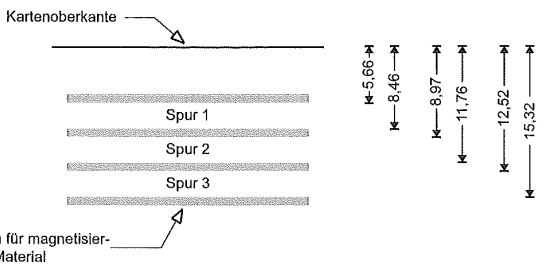


Bild 2.3 Die Lage der einzelnen Spuren auf der ID-1 Karte. Alle Maße sind in Millimeter angegeben.

Der Hauptnachteil der Magnetstreifentechnik besteht darin, dass die gespeicherten Daten sehr leicht verändert werden können. Während die Manipulation an hochgeprägten Zeichen zumindest einiges handwerkliches Geschick erfordert und von einem geschulten Auge auch leicht entdeckt werden kann, ist die Veränderung der auf der Magnetpiste kodierten Daten mit einem handelsüblichen Schreib-/Lesegerät problemlos möglich und später nur schwer nachweisbar. Hinzu kommt noch, dass die Magnetstreifenkarte häufig in Automaten zum Einsatz kommt, an denen eine visuelle Echtheitsprüfung unmöglich ist, wie z. B. in Geldausgabeautomaten. Der potenzielle Betrüger, der sich gültige Kartendaten verschafft hat, kann an solchen unbemannten Automaten einfache Duplikate von Karten verwenden, ohne dass er die visuellen Sicherheitsmerkmale fälschen muss.

Die Hersteller von Magnetstreifenkarten haben verschiedene Techniken entwickelt, um die Daten auf der Magnetpiste gegen Verfälschen und Duplizieren zu schützen. So sind z. B. die deutschen Eurocheque-Karten mit einer unsichtbaren und unveränderbaren Kodierung im Kartenkörper versehen, die eine Veränderung der Daten auf der Magnetpiste sowie ein Duplizieren unmöglich macht. Dieses und andere Verfahren erfordern allerdings einen speziellen Sensor im Kartenterminal, wodurch deren Kosten deutlich erhöht werden. Deshalb konnte sich bisher keines dieser Verfahren international durchsetzen.

Tabelle 2.1 Die einzelnen Spuren einer Magnetstreifenkarte mit ihren üblichen Eigenschaften nach ISO 7811.

Eigenschaft	Spur 1	Spur 2	Spur 3
Anzahl der Daten	max. 79 Zeichen	max. 40 Zeichen	max. 107 Zeichen
Codierung der Daten	6 Bit alphanumerischer Code	4 Bit BCD Code	4 Bit BCD Code
Datendichte	210 Bit/inch = 8,3 Bit/mm	75 Bit/Inch = 3 Bit/mm	210 Bit/inch = 8,3 Bit/mm
schreibbar	nicht erlaubt	nicht erlaubt	erlaubt

2.3 Chipkarten

Die Chipkarte ist das jüngste und schlaueste Kind in der Familie der Identifikationskarten im ID-1 Format. Es zeichnet sich dadurch aus, dass im Kartenkörper eine integrierte Schaltung versteckt ist, die über Elemente zur Datenübertragung, zum Speichern von Daten und zur Verarbeitung von Daten verfügt. Die Datenübertragung kann dabei entweder über die Kontakte an der Oberfläche der Karte erfolgen oder aber kontaktlos durch elektromagnetische Felder.

Die Chipkarte bietet gegenüber der Magnetstreifenkarte eine Reihe von Vorteilen. So ist z. B. die maximale Speicherkapazität von Chipkarten um ein Vielfaches größer als bei Magnetstreifenkarten. Heute werden bereits Schaltkreise mit mehr als 256 kByte Speicher angeboten, und dieser Wert wird sich mit jeder neuen Chipgeneration noch vervielfachen. Nur die im nächsten Kapitel beschriebenen optischen Speicherkarten haben eine noch größere Kapazität.

Einer der wichtigsten Vorteile der Chipkarte liegt jedoch darin, dass die in ihr gespeicherten Daten gegen unerwünschten Zugriff und gegen Manipulation geschützt werden können. Da der Zugriff auf die Daten nur über eine serielle Schnittstelle erfolgt, die vom Betriebssystem und einer Sicherheitslogik gesteuert wird, ist es möglich, geheime Daten in die Karte zu laden, die niemals mehr von außen gelesen werden können. Diese geheimen Daten können dann nur noch intern vom Rechenwerk des Chips verarbeitet werden. Grundsätzlich können die Speicherfunktionen Schreiben, Löschen und Lesen sowohl per Hardware als auch per Software eingeschränkt und an bestimmte Bedingungen geknüpft werden. Dies ermöglicht die Konstruktion einer Vielzahl von Sicherheitsmechanismen, die auf die speziellen Anforderungen der jeweiligen Anwendung zugeschnitten werden können.

Zusammen mit der Fähigkeit, Kryptoalgorithmen zu rechnen, ermöglicht die Chipkarte die Realisierung eines handlichen Sicherheitsmoduls, welches jederzeit – z. B. in der Brieftasche – mitgenommen werden kann.

Weitere Vorteile der Chipkarten liegen in der hohen Zuverlässigkeit und Lebensdauer im Vergleich zur Magnetstreifenkarte, deren Umlaufzeit im Allgemeinen auf ein bis zwei Jahre begrenzt ist.

Die wesentlichen Eigenschaften und Funktionen von Chipkarten sind in den ISO-Normen der Reihe 7816 festgelegt. Wir werden in den folgenden Kapiteln noch ausführlich auf diese Standards eingehen.

Wegen der Unterschiede in der Funktionalität, aber auch im Preis, werden die Chipkarten in Speicherkarten und Mikroprozessorkarten eingeteilt.

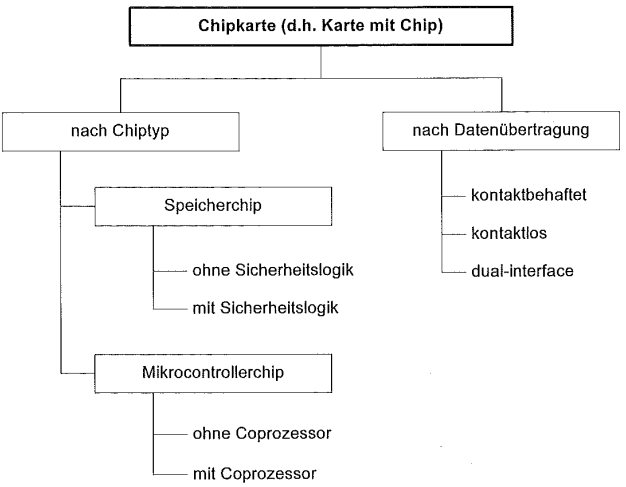


Bild 2.4 Klassifizierungsbaum der Karten mit einem Chip differenziert nach Chiptyp und nach Datenübertragung.

2.3.1 Speicherkarten

Die Architektur einer Speicherkarte ist im Bild 2.5 als Blockschaltbild dargestellt. Im Speicher – meist ein EEPROM – werden die für die Anwendung erforderlichen Daten abgelegt. Der Zugriff auf den Speicher wird durch die Sicherheitslogik kontrolliert, welche im einfachsten Fall nur aus einem Schreib- oder Löschschutz für den Speicher oder einzelnen Bereichen des Speichers besteht. Es gibt aber auch Speicherchips mit einer komplexeren Sicherheitslogik, die auch einfache Verschlüsselungen durchführen können. Über das I/O-Port werden die Daten von und zur Karte übertragen. Hierzu ist in Teil 3 von ISO 7816 ein spezielles synchrones Übertragungsprotokoll definiert, welches eine beson-

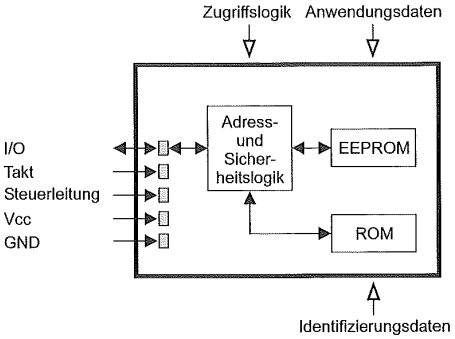


Bild 2.5 Typische Architektur einer kontaktbehafteten Speicherkarte mit Sicherheitslogik. Die Abbildung zeigt lediglich die grundlegenden Informations- und Energieflüsse und ist kein Stromlaufplan.

ders einfache und damit preiswerte Realisierung im Chip ermöglicht. Es kommt aber auch der bei Speichern mit seriellem Zugriff weit verbreitete I^2C-Bus in Chipkarten zum Einsatz.

Die Funktionalität der Speicherkarten ist meist auf eine spezielle Anwendung hin optimiert. Hierdurch ist die Flexibilität in der Anwendung zwar stark eingeschränkt, dafür sind Speicherkarten aber auch besonders preisgünstig. Typische Anwendungen für Speicherkarten sind vorbezahlte Telefonkarten oder die Krankenversicherungskarte.

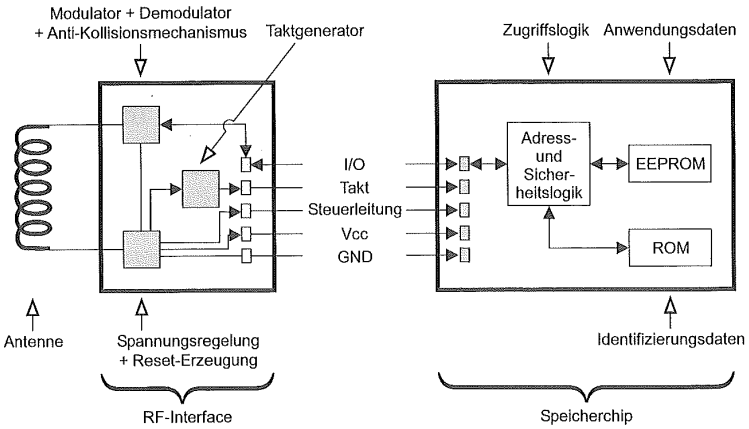


Bild 2.6 Typische Architektur einer Speicherkarte mit kontaktlosem Interface und Sicherheitslogik. Die Abbildung zeigt lediglich die grundlegenden Informations- und Energieflüsse und ist kein Stromlaufplan.

2.3.2 Mikroprozessorkarten

Das Herz des Chips einer Mikroprozessorkarte ist – wie der Name schon sagt – der Prozessor, der in der Regel von vier weiteren Funktionsblöcken umgeben ist: dem Masken-ROM, dem EEPROM, dem RAM und dem I/O-Port. In Bild 2.7 ist die typische Architektur eines solchen Bausteins dargestellt.

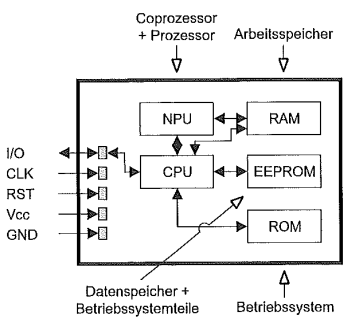


Bild 2.7 Typische Architektur einer kontaktbehafteten Mikroprozessorkarte mit Koprozessor. Die Abbildung zeigt lediglich die grundlegenden Informations- und Energieflüsse und ist kein Stromlaufplan.

Das Masken-ROM enthält das Betriebssystem des Chips und wird während der Herstellung eingebrannt. Der Inhalt des ROM ist herstellungsbedingt für alle Chips eines Produktionsloses identisch und während der Lebensdauer des Chips unveränderbar.

Das EEPROM ist der nichtflüchtige Speicherbereich des Chips, in dem Daten oder auch Programmcode unter Kontrolle des Betriebssystems geschrieben und gelesen werden können.

Das RAM ist der Arbeitsspeicher des Prozessors. Dieser Speicherbereich ist flüchtig, und alle darin gespeicherten Daten gehen verloren, wenn die Versorgungsspannung des Chips abgeschaltet wird.

Die serielle I/O-Schnittstelle besteht meist nur aus einem einzigen Register, über welches die Daten Bit für Bit übertragen werden.

Mikroprozessorkarten sind in der Anwendung sehr flexibel. Im einfachsten Fall enthalten sie ein auf eine einzige Anwendung hin optimiertes Programm und sind somit auch nur für diese eine Anwendung verwendbar.

Moderne Chipkartenbetriebssysteme ermöglichen jedoch, verschiedene Anwendungen in einer einzigen Karte zu integrieren. Das ROM enthält in diesem Falle nur die Basiskommandos des Betriebssystems, während die anwendungsspezifischen Teile des Programms erst nach der Kartenproduktion in das EEPROM geladen werden. Neue Entwicklungen ermöglichen es sogar, Anwendungen in die Chipkarte nachzuladen, nachdem die Karte personalisiert und an den Kartenbenutzer ausgegeben wurde. Durch entsprechende Hardware- und Softwaremaßnahmen wird hierbei sichergestellt, dass die unterschiedlichen Sicherheitsbedingungen der einzelnen Anwendungen hierbei nicht verletzt werden. Speziell hierfür optimierte Mikroprozessorchips mit hoher Rechenleistung und Speicherkapazität sind mittlerweile verfügbar.

2.3.3 Kontaktlose Chipkarten

Die Kontaktierung der kontaktbehafteten Chipkarte erfolgt über die acht in der ISO-Norm 7816 Teil 1 festgelegten Kontakte. Die Zuverlässigkeit der Chipkarten mit Kontakten konnte aufgrund der steigenden Produktionserfahrung der Hersteller in den vergangenen Jahren stetig verbessert werden, sodass zum Beispiel die Ausfallquote von Telefonkarten über eine Lebensdauer von einem Jahr heute deutlich unter ein Promille liegt. Nach wie vor sind jedoch Kontakte eine der häufigsten Fehlerquellen in elektromechanischen Systemen. Störungen können zum Beispiel durch Verschmutzung oder Abnutzung der Kontakte entstehen. Beim Einsatz in mobilen Geräten können Vibrationen zu kurzzeitigen Kontaktunterbrechungen führen. Da die Kontakte auf der Oberfläche der Chipkarte direkt mit den Eingängen der integrierten Schaltung verbunden sind, besteht die Gefahr, dass elektrostatische Entladungen – einige tausend Volt sind durchaus keine Seltenheit – die integrierte Schaltung im Innern der Karte schwächen oder gar zerstören.

Diese technischen Probleme werden von der kontaktlosen Chipkarte elegant umgangen.

Außer diesen technischen Vorteilen bietet die Technik der kontaktlosen Chipkarte aber auch eine Reihe interessanter neuer Möglichkeiten in der Anwendung für den Kartenherausgeber und den Kartenbenutzer. So müssen kontaktlose Chipkarten zum Beispiel nicht unbedingt in einen Kartenleser eingesteckt werden, sondern es gibt Systeme, die über eine Entfernung von bis zu einem Meter funktionieren. Dies ist beispielsweise in Zugangskon-

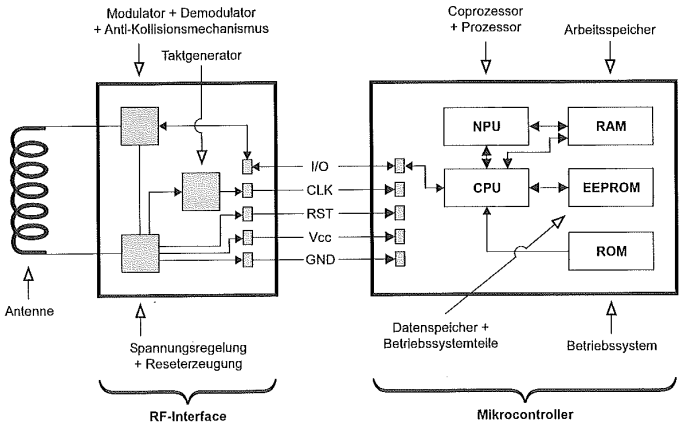


Bild 2.8 Typische Architektur einer Mikroprozessorkarte mit Koprozessor und kontaktlosem Interface. Die Abbildung zeigt lediglich die grundlegenden Informations- und Energieflüsse und ist kein Stromlaufplan.

trollsystemen, bei denen eine Tür oder ein Drehkreuz geöffnet werden soll, ein großer Vorteil, da die Zugangsberechtigung einer Person geprüft werden kann, ohne dass diese die Karte aus der Tasche nehmen und in ein Kartenlesegerät einstecken muss. Ein großes Anwendungsgebiet ist hierfür der öffentliche Personennahverkehr, wo in möglichst kurzer Zeit möglichst viele Personen erfasst werden müssen.

Aber auch in Systemen, in denen das bewusste Einführen der Karte in das Lesegerät ausdrücklich verlangt wird, bietet die kontaktlose Technik den Vorteil, dass die Orientie-

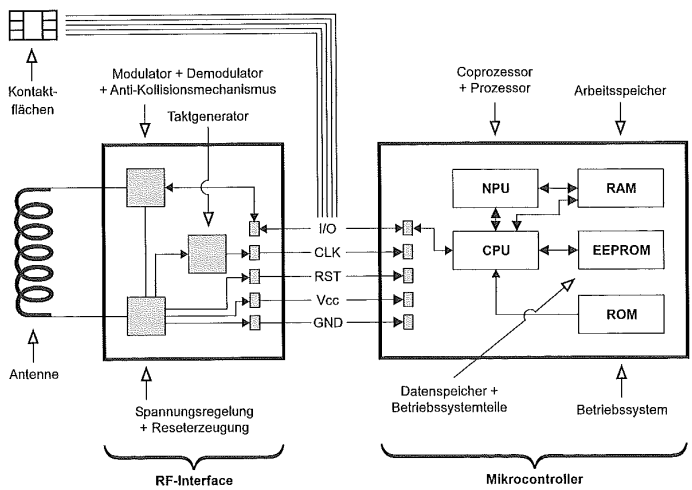


Bild 2.9 Typische Architektur einer Mikroprozessorkarte mit Koprozessor, kontaktlosem sowie kontaktbehaftetem Interface. Die Abbildung zeigt lediglich die grundlegenden Informations- und Energieflüsse und ist kein Stromlaufplan.

rung der Karte beim Einstecken in den Leser beliebig sein kann, im Gegensatz zur Magnetstreifenkarte oder kontaktbehafteten Karte, die nur in einer ganz bestimmten Lage funktioniert. Das erleichtert die Handhabung wesentlich und erhöht somit die Akzeptanz beim Kunden.

Eine weitere interessante Variante der Anwendung von kontaktlosen Karten ist die Benutzung an der Oberfläche des Lesers. Hierbei wird die Karte nicht in einen Schlitz gesteckt, sondern einfach auf eine markierte Stelle auf der Oberfläche des Kartenlesers gelegt. Neben einfacher Handhabung besticht diese Lösung auch dadurch, dass die Gefährdung durch Vandalismus (typische Beispiele: Kaugummi oder Sekundenkleber im Kartenschlitz) deutlich geringer ist.

Für die Vermarktung der Karten bietet die kontaktlose Technik den Vorteil, dass keine technischen Elemente an der Kartenoberfläche sichtbar sind, sodass die optische Gestaltung der Kartenoberfläche nicht durch Magnetstreifen oder Kontaktflächen eingeschränkt wird. Diese Vorteile werden allerdings durch den Nachteil aufwendiger Terminals mit einem entsprechend höheren Preis erkauft. Ein weiterer Nachteil besteht darin, dass mehrere verschiedene Systeme für kontaktlose Karten standardisiert und im Markt sind. Hierdurch wird die Komplexität eines Terminals, das zu allen genormten Karten kompatibel sein soll, weiter erhöht.

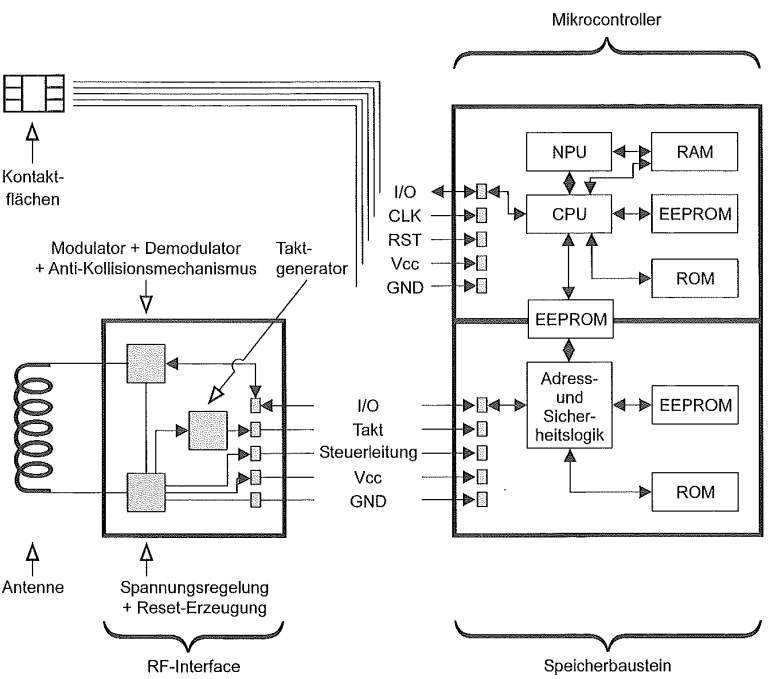


Bild 2.10 Typische Architektur einer Dual-Interface-Karte, die eine kontaktlose Speicherkarte mit einer kontaktbehafteten Mikroprozessorkarte kombiniert. Die Abbildung zeigt lediglich die grundlegenden Informations- und Energieflüsse und ist kein Stromlaufplan.

Die Produktionstechnik für die Massenfertigung von kontaktlosen Karten ist inzwischen so weit ausgereift, dass qualitativ hochwertige Produkte zu Preisen angeboten werden, die nur unwesentlich höher sind als bei vergleichbaren kontaktbehafteten Erzeugnissen. Die häufigsten Anwendungen finden die kontaktlosen Chipkarten bisher im öffentlichen Personennahverkehr, wo sie als elektronisches Ticket ein modernes elektronisches Fahrgeldmanagement ermöglichen. Während die gegenwärtig betriebenen Systeme in der Regel noch monofunktionale Karten verwenden, wozu preisgünstige Speicherchips mit festverdrahteter Sicherheitslogik entwickelt wurden, zeichnet sich zunehmend der Bedarf ab, das elektronische Ticket mit einer zusätzlichen Bezahlfunktion zu versehen. Es werden deshalb vermehrt Multifunktionskarten mit integriertem Mikroprozessor zum Einsatz kommen, wobei die Bezahlfunktion meist noch über die konventionelle Technik mit Kontakten erfolgt, um die vorhandene Infrastruktur z. B. der elektronischen Geldbörsensysteme mit benutzen zu können. Diese Karten verfügen sowohl über Kontakte wie auch über kontaktlose Koppelelemente und werden als Dual-Interface-Karten oder auch Combikarten bezeichnet.

Eine ausführliche Beschreibung der Technik und Funktionsweise der kontaktlosen Chipkarten findet sich in Abschnitt „3.6 Kontaktlose Karten".

2.4 Optische Speicherkarten

Für Anwendungen, bei denen die Speicherkapazität der Chipkarten nicht ausreicht, bietet sich die optische Speicherkarte an, auf der mehrere MByte an Daten Platz finden. Bei den heute verfügbaren Techniken können die Karten allerdings nur ein einziges Mal beschrieben und nicht wieder gelöscht werden.

Die Normen ISO/IEC 11693 und 11694 definieren die physikalischen Eigenschaften und die Technik für die lineare Datenaufzeichnung der optischen Speicherkarten.

Interessante Aspekte ergeben sich, wenn man die hohe Speicherkapazität der optischen Speicherkarte mit der Intelligenz der Chipkarte kombiniert. Man kann dann z. B. die Daten in verschlüsselter Form in den optischen Speicher schreiben und den Schlüssel im geheimen Speicher des Chips sicher aufbewahren. Hierdurch können die optisch gespeicherten

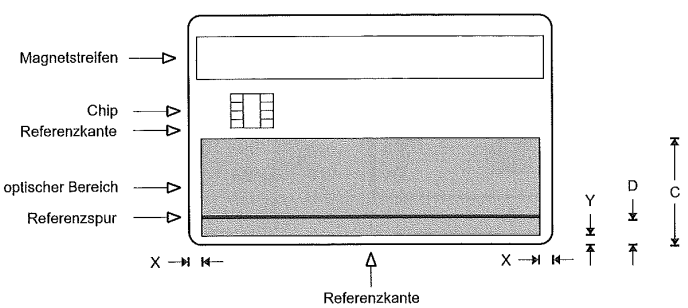


Bild 2.11 Anordnung des optischen Bereiches nach ISO/IEC 11 694-2 bei einer ID-1 Karte.
C: 9,5 mm ... 49,2 mm, D: 5,8 mm ± 0,7 mm
X: bei PWM: maximal 3 mm; bei PPM: maximal 1 mm
Y: bei PWM: Y<D und minimal 1 mm; bei PPM: maximal 4,5 mm
(*PWM: pulse width modulation; PPM: pulse position modulation*)

Daten vor unerlaubtem Zugriff geschützt werden. In Bild 2.11 ist eine typische Ausführung einer Chipkarte mit Kontakten, Magnetstreifen und optischer Speicherfläche dargestellt. Man sieht, dass die verfügbare optische Speicherfläche durch die Chipkontakte eingeschränkt wird, was natürlich auch die maximale Speicherkapazität begrenzt. Der Magnetstreifen befindet sich auf der Rückseite der Karte.

Die Schreib- und Lesegeräte für optische Speicherkarten sind zurzeit noch sehr teuer, was ihre Anwendungsmöglichkeiten bisher sehr eingeschränkt hat.

Eingesetzt werden optische Karten z. B. im Gesundheitswesen zur Speicherung von Patientendaten, wobei die hohe Speicherkapazität sogar die Speicherung von Röntgenbildern auf der Karte ermöglicht.

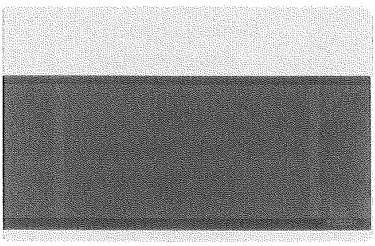


Bild 2.12 Typische optische Speicherkarte mit einer Nettospeicherkapazität, d. h. mit Fehlerkorrektur von ca. 4 MByte. Die Bruttospeicherkapazität, d. h. ohne Fehlerkorrektur, bewegt sich in der Größenordnung von 6 MByte.

Bild 2.13 Karte, die von einem CD-Laufwerk gelesen werden kann und eine Speicherkapazität von ca. 32 MByte besitzt.

3 Physikalische und elektrische Eigenschaften

Die grundlegenden Eigenschaften des Kartenkörpers von Chipkarten stammen von den Vorgängern dieser Karten. Dies sind die seit langem bekannten Prägekarten, die im Kreditkartensektor momentan immer noch marktbeherrschend sind. Technisch gesehen sind sie einfach aufgebaute Karten aus Kunststoff, die durch Einprägung von diversen Benutzermerkmalen wie Name und Kundennummer personalisiert sind.

In weiterentwickelten Ausführungen wurden die Karten mit einem Magnetstreifen versehen, der die einfache maschinelle Verarbeitung ermöglicht. Als dann die Idee aufkam, Karten mit einem Chip auszustatten, verwendete man die ursprünglichen Karten als Grundlage und implantierte zusätzlich noch einen Mikrocontroller. Viele Normen über die physikalischen Eigenschaften einer Karte sind deshalb nicht spezifisch für Chipkarten, sondern decken ebenfalls auch noch Magnetstreifen- und Prägekarten ab.

3.1 Physikalische Eigenschaften

Nimmt man eine Chipkarte in die Hand, dann ist das unmittelbar auffallende Merkmal das Format der Karte. Danach sieht man vielleicht, ob die Karte mit einer Kontaktfläche ausgestattet ist, obwohl bei kontaktlosen Chipkarten überhaupt keine sichtbare elektrische

Schnittstelle vorhanden ist. Magnetstreifen, Hochprägung und Hologramm fallen eventuell als Nächstes ins Auge. Alle diese Merkmale und Funktionsteile sind Teil der physikalischen Eigenschaften einer Chipkarte.

Ein Großteil der physikalischen Eigenschaften sind in Wahrheit rein mechanischer Natur, wie Größe, Biege- und Torsionsfestigkeit. Diese kennt man auch im praktischen Umgang mit Chipkarten unmittelbar aus der Erfahrung. In der Praxis spielen jedoch sehr wohl typische physikalische Eigenschaften wie Temperatur-, Lichtempfindlichkeit oder Feuchtebeständigkeit eine Rolle.

Es ist auch immer das Zusammenwirken von Kartenkörper und implantiertem Chip zu betrachten, da nur beide miteinander eine funktionsfähige Chipkarte bilden. Ein Kartenkörper, der beispielsweise für hohe Umgebungstemperatur geeignet ist, nützt wenig, wenn dies nicht auch der Mikrocontroller ist. Beide Teile müssen sowohl einzeln als auch miteinander alle notwendigen Anforderungen erfüllen, da es sonst zu hohen Ausfallraten beim Einsatz kommen kann.

3.1.1 Formate

Es wurden schon sehr lange kleine Kärtchen in der typischen Chipkartengröße von 85,6 mm Länge und 54 mm Breite verwendet. In diesem Format, das mit Sicherheit auch das bekannteste ist, werden fast alle Chipkarten produziert. Es hat die Bezeichnung ID-1 und ihre Größe ist in der internationalen Norm ISO 7810 festgelegt. Anhand der Abkürzung „ID", die für Identifikationskarte steht, sieht man deutlich, dass diese erste Norm aus dem Jahre 1985 noch nichts mit den Chipkarten, wie wir sie heute kennen, zu tun hatte. Dort wird lediglich eine Kunststoffkarte mit einem Magnetstreifen und Hochprägung beschrieben, die zur Identifizierung von Personen dient. An einen in die Karte eingebauten Chip war damals noch nicht gedacht. Erst einige Jahre später definierte man in weiteren Normen das Vorhandensein eines Chips und die Position seiner Kontaktflächen auf der Karte.

Bei der heutigen Vielfalt an Karten, die für alle möglichen Zwecke verwendet werden und die verschiedenste Abmessungen haben, ist es mittlerweile oft schwierig festzustellen,

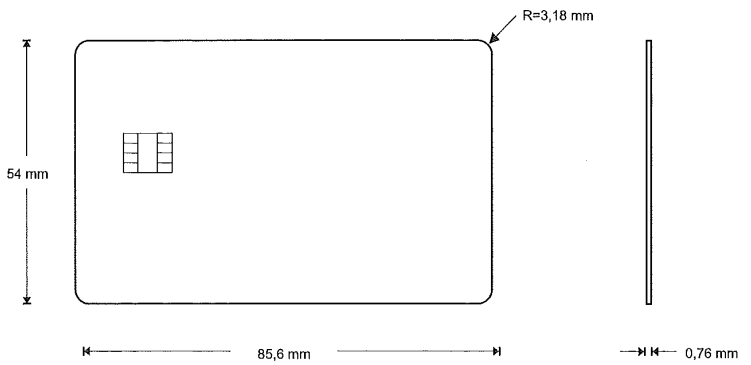


Bild 3.1 Das ID-1 Format nach ISO 7810. Die Karte hat eine Dicke von 0,76 mm ± 0,08 mm und der Radius der Ecken beträgt 3,18 mm ± 0,30 mm. Die eingezeichneten Abmessungen stellen die Maße ohne Toleranzen dar.

ob eine Karte nun eigentlich eine ID-1 Chipkarte ist oder nicht. Eines der besten Erkennungsmerkmale ist neben dem implantierten Chip die Dicke der Karte. Beträgt die Dicke einer Karte mit implantiertem Mikrocontroller 0,76 mm, kann man im eigentlichen Sinne von einer Chipkarte nach ISO–Norm sprechen.

Das gebräuchliche ID-1 Format hat den Vorteil, dass es manuell sehr gut handhabbar ist. Das Format der Karte ist so festgelegt, dass man sie in einer üblichen Geldbörse noch mit sich führen kann, und nicht zu klein, um sie leicht zu verlieren. Außerdem ist sie wegen ihrer Flexibilität weniger störend als ein steifer Gegenstand.

Allerdings entspricht das Format oft nicht mehr den Erfordernissen der heutigen Miniaturisierung. Tragbare Telefone sind zum Teil leichter als 100 g und so groß wie eine Packung Papiertaschentücher. Deshalb war es notwendig, neben ID-1 ein kleineres Format zu definieren, das die Belange der kleinen Endgeräte berücksichtigt. Die Karte durfte dabei sehr klein sein, da sie üblicherweise nur ein einziges Mal in das Mobiltelefon gesteckt werden muss und dann für immer dort bleibt. Unter diesen Voraussetzungen definierte man das ID-000 Format, dessen einprägsamere Bezeichnung „Plug-In" oder deutsch „Einschubkarte" lautet. Dieses Format kommt momentan nur in Endgeräten des Mobilfunkbereichs zum Einsatz.

Da jedoch die Handhabung der ID-000 Karte sowohl in der Fertigung als auch bei den Benutzern nicht gerade einfach ist, führte dies zu einem weiteren Format. Benannt ist dieses Format als ID-00 bzw. „Mini-Karte". Es stellt in der Größe ungefähr den Mittelwert zwischen ID-1 und ID-000 dar. Auf diese Weise lässt sich die Karte einfacher manuell handhaben und ermöglicht auch eine kostengünstige Fertigung, weil sie sich durch Ausstanzen aus dem ID-1 Format herstellen lässt. Die Definition von ID-00 gibt es bereits seit Mitte der 90er-Jahre, doch hat sich dieses Kartenformat bis jetzt weder national noch international etabliert.

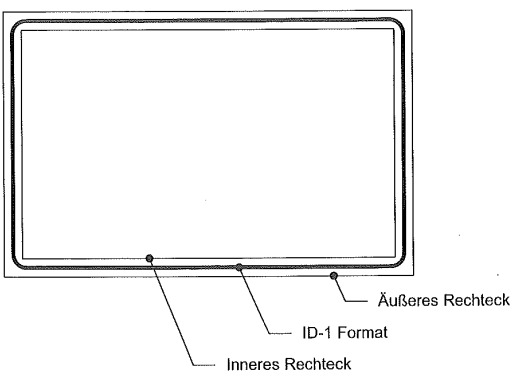


Bild 3.2 Die Definition der Maße des ID-1 Formats für Chipkarten nach GSM 11.11 und TS 102.221.

Die Definition der Formate in den jeweiligen Normen wurde auf messtechnische Belange hin optimiert. So muss ein Kartenkörper im Format ID-1 in seiner Höhe und Breite so beschaffen sein, dass er ohne die abgerundeten Ecken innerhalb zweier konzentrischer und symmetrisch zueinander angeordneter Rechtecke mit den folgenden Abmessungen passt:

Äußeres Rechteck:	Breite	85,72 mm (= 3,375 inch)
	Höhe	54,03 mm (= 2,127 inch)
Inneres Rechteck:	Breite	85,46 mm (= 3,365 inch)
	Höhe	53,92 mm (= 2,123 inch)

Die Dicke muss 0,76 mm (= 0,03 inch) mit einer Toleranz von ± 0,08 mm (= ± 0,003 inch) betragen. Die Radien der Ecken und die Dicke des Kartenkörpers sind auf konventionelle Art bemaßt. Auf der Grundlage dieser Definitionen kann man die Abmessungen einer ID-1 Karte in dem Bild 3.1 darstellen.

Dem ID-000 Format nach GSM 11.11 und TS 102.221 dienen ebenfalls zwei konzentrische Rechtecke als Grundlage der Formatdefinition. Da dieses Format seine Ursprünge in Europa hat (Grundlage war das Mobilfunksystem GSM), basiert es auf metrischen Grundmaßen. Die Abmessungen der beiden Rechtecke sind:

Äußeres Rechteck:	Breite	25,10 mm
	Höhe	15,10 mm
Inneres Rechteck:	Breite	24,90 mm
	Höhe	14,90 mm

Die rechte untere Ecke des Plug-In ist im Winkel von 45° abgetrennt, um damit eine eindeutige Orientierung der Karte beim Einlegen in den Kartenleser sicherzustellen.

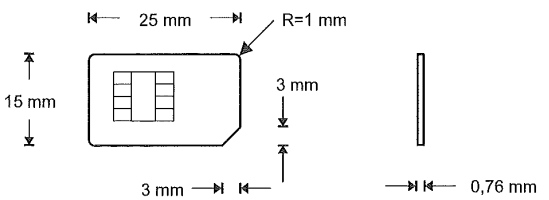


Bild 3.3 Das ID-000 Format. Die Karte hat eine Dicke von 0,76 mm ± 0,08 mm, der Radius der Ecken beträgt 1 mm ± 0,10 mm und die Ecke 3 mm ± 0,03 mm. Die eingezeichneten Abmessungen stellen die Maße ohne Toleranzen dar.

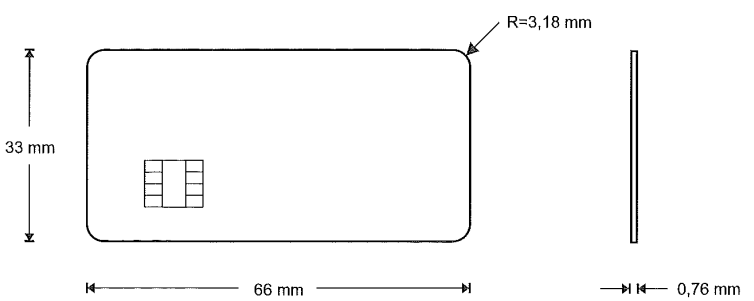


Bild 3.4 Das ID-00 Format. Die Karte hat eine Dicke von 0,76 mm ± 0,08 mm und der Radius der Ecken beträgt 3,18 mm ± 0,30 mm. Die eingezeichneten Abmessungen stellen die Maße ohne Toleranzen dar.

Das ebenfalls auf metrischen Grundmaßen basierende ID-00 Format ist gleichfalls in seinen Maximal- und Minimalgrößen durch zwei konzentrische Rechtecke definiert. Die Abmessungen dieser beiden Rechtecke sind:

Äußeres Rechteck:	Breite	66,10 mm
	Höhe	33,10 mm
Inneres Rechteck:	Breite	65,90 mm
	Höhe	32,90 mm

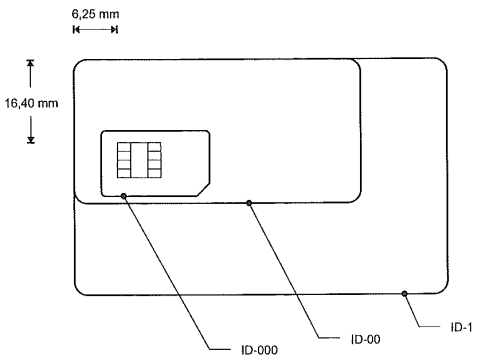


Bild 3.5 Die Zusammenhänge zwischen den Kartenformaten ID-1, ID-00 und ID-000.

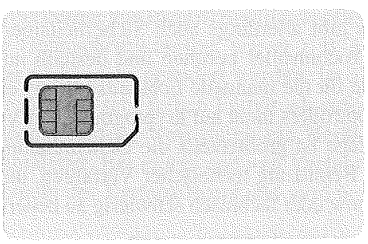


Bild 3.6 Beispiel für eine Mobilfunkkarte im ID-1 Format, die vom Kunden bei Bedarf durch Ausbrechen in das ID-000 überführt werden kann.

Die drei Formate ID-1, ID-00 und ID-000 können von den größeren Kartenkörpern zu den kleineren durch Stanzen überführt werden. Dies ist vor allem für die Kartenhersteller interessant, da sich damit der Fertigungsprozess sehr gut optimieren und kostengünstiger im einheitlichen ID-1 Format gestalten lässt.

So ist es zum Beispiel üblich, dass ein Hersteller nur Karten in einem einzigen Format (vorzugsweise ID-1) produziert, dann die Module einbettet und komplett personalisiert. Abhängig vom genauen Einsatzgebiet der solchermaßen hergestellten Chipkarten können diese dann in einem nachgeordneten Produktionsschritt auf das gewünschte Format gebracht werden.

Alternativ dazu kann dies auch der Kunde machen, wie es im Mobilfunkbereich seit langem Usus ist. Der Kunde erhält eine ID-1 Karte, die vorgestanzt ist, sodass man durch Ausbrechen eine ID-000 Karte erhält. In einem anderen Verfahren ist die ID-000 komplett aus der ID-1 ausgestanzt und auf der den Kontakten abgewandten Seite durch ein einseitiges Klebeband mit dem restlichen Kartenkörper der ID-1 Karte verbunden. Somit kann der

Kunde je nach vorhandenem Endgerät sein Kartenformat selbst herstellen, und die Produktion und der Versand bleiben beim Kartenhersteller ohne Varianten.

Für manche Anwendungen ist das typische Kartenformat jedoch mit einigen Nachteilen verbunden. Als Lösung bietet sich hier die Verwendung anderer Formfaktoren, wie beispielsweise USB-Stecker, mit integriertem Chipkarten-Mikrocontroller an. Das logische Verhalten dieser Chipkarten-Varianten entspricht in aller Regel vollständig dem bei den gängigen Formen.

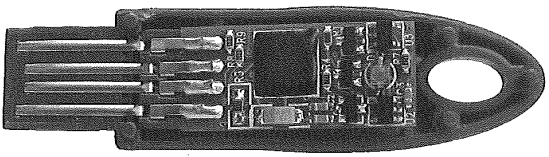


Bild 3.7 Beispiel für einen anderen Formfaktor einer Chipkarte. Das Bild zeigt einen geöffneten USB-Stecker mit einem eingelöteten Chipkarten-Mikrocontroller sowie einigen notwendigen Schnittstellenkomponenten.

3.1.2 Kartenelemente und Sicherheitsmerkmale

Da Chipkarten meist als Berechtigungen für bestimmte Dinge oder zur Identifizierung des Inhabers genutzt werden, sind oft zusätzlich zu dem implantierten Chip Sicherheitsmerkmale auf dem Kartenkörper erforderlich. Da eine Kartenechtheitsprüfung auch durch Menschen und nicht nur durch Maschinen durchgeführt wird, beruhen viele Sicherheitsmerkmale auf optischen Merkmalen. Es gibt allerdings auch Sicherheitsmerkmale, die auf einem modifizierten Chipkarten-Mikrocontroller beruhen und deshalb nur durch Computer prüfbar sind. Die üblichen Merkmale für die manuelle Echtheitsprüfung einer Karte basieren im Gegensatz zu denen des Mikrocontrollers nicht auf kryptografischen Verfahren (z. B.: gegenseitige Authentisierung). Grundlage ist größtenteils die Geheimhaltung von Materialien und Herstellungsverfahren oder die Benutzung technischer Verfahren, die nur mit sehr großem Aufwand, erheblichem Know-how oder technisch schwierig zu beherrschen sind.

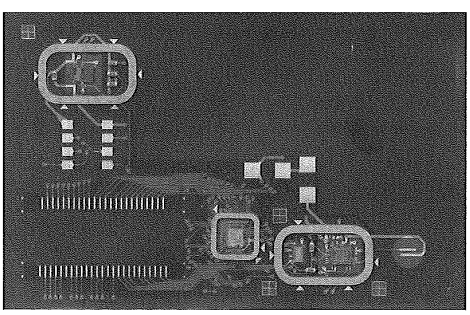


Bild 3.8 Inlayfolie für eine Super Smart Card. Links oben sieht man den eigentlichen Chipkarten-Mikrocontroller, der mit den darunter liegenden Kontaktflächen verbunden ist. Unten links befinden sich die Anschlussstellen für ein Display. Rechts daneben ist die Ansteuerschaltung für das Display und darüber vier große Kontakte für eine Batterie zu sehen. Rechts unten kann man die Kontakte für einen Druckschalter erkennen und links daneben sind verschiedene Komponenten zur Schnittstellenanpassung platziert. (Giesecke & Devrient)

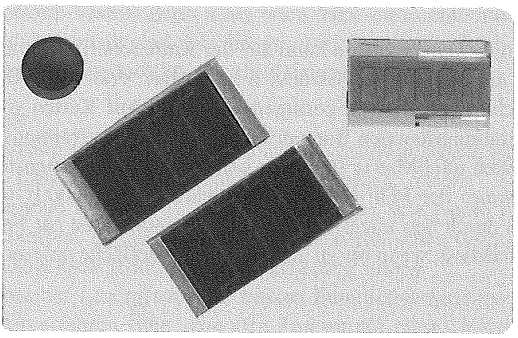


Bild 3.9 Frühes Labormuster einer Super Smart Card für eine kontaktlos betriebene elektronische Geldbörse. Links oben sieht man einen Druckschalter zur Bestätigung der Transaktionen und in der Mitte zwei Solarzellen zur Stromversorgung. Rechts oben befindet sich ein fünfstelliges Display zur Anzeige von Börsensaldo und anderen Daten. (Giesecke & Devrient)

Gerade im Bereich neuer Kartenelemente besteht für die nähere Zukunft ein erhebliches Entwicklungspotenzial, da eine mögliche Evolutionsrichtung von Chipkarten in Richtung zusätzlich integrierter Bauteile wie Tastatur, Display, Solarzelle und Batterie führt.

Unterschriftsstreifen
Ein eher einfaches Verfahren zur Identifizierung des Besitzers ist ein fest mit dem Kartenkörper verbundener Unterschriftsstreifen, wie er z. B. auf Kreditkarten üblich ist. Einmal beschrieben, kann er nicht mehr geändert werden, ist also radierfest. Ein Überkleben des Streifens würde durch einen sehr feinen Farbdruck sofort sichtbar. Die irreversible Befestigung des Unterschriftsstreifens wird durch ein Heißklebeverfahren für den bedruckten Papierstreifen gewährleistet. Bei einem anderen Verfahren ist der Unterschriftsstreifen ein Teil der obersten Schicht des Kartenkörpers und wird beim Zusammenbau einlaminiert.

Guillochen
Etwas aufwendiger sind unter der transparenten obersten Schicht der Karte eingefügte Folien mit farbig aufgedruckten Guillochen. Als Guillochen werden die meist runden oder ovalen geschlossenen und miteinander verwobenen Linienfelder bezeichnet, die sich auch auf manchen Geldscheinen oder Aktien befinden. Diese Muster können aufgrund ihrer feinen Struktur momentan nur auf drucktechnischem Wege erzeugt werden und sind deshalb ebenfalls nur schwer kopierbar.

Mikroschrift
Ein weiteres Verfahren, das ebenfalls auf der Sicherheit von feinen gedruckten Strukturen basiert, sind Mikroschrift-Linien. Diese für das Auge nur als Linie erkennbare Schrift kann nur mit einer Lupe erkannt werden und ist auch nicht kopierbar.

UV-Schrift
Um das sichtbare Kartenlayout nicht zu beeinflussen, können Kontrollzeichen oder Kontrollnummern mit einer nur unter UV-Licht sichtbaren Farbe auf den Kartenkörper aufgedruckt werden.

Barcode
Zur Speicherung von geringen Datenmengen kann auf der Kartenoberfläche mit Laserung oder Thermotransferdruck ein Barcode aufgedruckt werden. Der Vorteil vor Barcodes, ist die automatisierbare optische Lesbarkeit in geringem Abstand. Auf Karten werden nicht nur die weit verbreiteten eindimensionalen Barcodes, sondern auch zweidimensionale Barcodes in Form von gestapelten oder Matrix Barcodes benutzt. Ein zweidimensionaler Matrix Barcode wie beispielsweise der PDF 417 kann durchaus bis zu 1000 Bytes codieren und erreicht im besten Fall durch den integriertem Reed-Solomon-Code zur Fehlerkorrektur noch eine Datenrekonstruktion, wenn bis zu 25 % der Barcodefläche unleserlich sind.

Hologramm
Ein mittlerweile allen Kartenbenutzern bekanntes Sicherheitsmerkmal sind in die Karte integrierte Hologramme. Die Sicherheit der Hologramme liegt vor allem darin, dass es weltweit nur sehr wenige Hersteller gibt und sie nicht an jeden abgegeben werden.

Die bei Chipkarten benutzten Hologramme sind so genannte Prägehologramme, da sie auch bei diffusem weißem Tageslicht in Draufsicht erkannt werden müssen. Deshalb heißt diese Art von Hologramm auch Weißlicht-Reflexionshologramm. Übliche Durchlichthologramme würden nämlich kohärentes Laserlicht benötigen, um auf ihnen Bilder zu sehen. Manchmal besitzen Hologramme noch zusätzliche integrierte Sicherheitsmerkmale, die nur mit einem Laser sichtbar gemacht werden können. Um Prägehologramme herzustellen, muss zuerst ein Masterhologramm in der üblichen Technik für holographische Aufnahmen erstellt werden. Aus diesem wird mit einem Transferprozess ein Mutterprägestempel mit darauf befindlicher Mikrostruktur des späteren Prägehologramms erzeugt. Auf galvanotechnischem Wege können nun Tochterprägestempel gefertigt werden, mit denen dann die Mikrostruktur in Kunststofffolien geprägt wird. Die Folien werden anschließend mit Aluminium bedampft, und man erhält die bekannten Weißlicht-Reflexionshologramme.

Hologramme werden irreversibel mit dem Kartenkörper verbunden, sodass sie sich nur mehr zerstörend von ihm lösen lassen. Dies ist entweder mit einem Laminierverfahren möglich oder auch durch das Roll-On-Verfahren. Dabei wird das auf einem Zwischenträger befindliche Hologramm mit einer beheizten Walze auf die Karte gepresst und die Trägerfolie nach diesem Heißsiegeln abgezogen. Das Hologramm wird dabei unwiderruflich mit dem Kunststoffkartenkörper verschweißt. Das dritte mögliche Verfahren heißt „Hot-Stamp-Verfahren" und verläuft ähnlich dem Roll-On-Verfahren, nur dass statt einer beheizten Walze ein beheizter Stempel benutzt wird.

Kinegramm, Kippbild
Ein Kinegramm, oft auch Kippbild genannt, ist vom Aufbau her ein Hologramm, dessen Abbildung sich sprunghaft mit dem Betrachtungswinkel ändert. Kinegramme sind ähnlich schwer zu fälschen wie Hologramme, haben aber den Vorteil, dass sie vom Betrachter schneller erkannt und somit geprüft werden können.

Multiple Laser Image (MLI)
Ähnlich einem Hologramm ist das Multiple Laser Image: eine Art Kippbild, das einfachen Hologrammen sehr ähnlich ist. Es basiert auf einem auf die Oberfläche einer Karte geprägten Linsensystem, in dem Teile mit einem Laser geschwärzt werden. Der Hauptunterschied

zum Hologramm besteht darin, dass beim MLI kartenindividuelle Informationen auf dem kleinen Bild dargestellt sind. Bei diesem Verfahren ließe sich beispielsweise der Name des Kartenbesitzers individuell als Kippbild anbringen.

Lasergravur

Die Schwärzung von speziellen Kunststoffschichten durch Verbrennen mit einem Laser nennt man Lasergravur oder schlicht und einfach „lasern". Dies ist im Gegensatz zum Hochprägen eine sichere Art, individuelle Daten, wie Name und Kartennummer, auf die Karte zu schreiben, da die dazu notwendigen technischen Gerätschaften und das dazugehörige Know-how nicht jedermann zur Verfügung stehen.

Es gibt bei der Lasergravur die beiden Funktionsprinzipien Vektor- und Rastergravur. Beim Vektorverfahren führt man den Laserstrahl ohne Unterbrechung einer Bahn nach. Dies eignet sich sehr gut für die Darstellung von Zeichen und hat den Vorteil einer hohen Geschwindigkeit. Im Gegensatz dazu werden bei der Lasergravur im Rasterverfahren, ähnlich einem Tintenstrahl- oder Nadeldrucker, sehr viele nebeneinander liegende Punkte geschwärzt. Einsatz findet dieses Verfahren vor allem bei der Aufbringung von Bildern auf Karten. Es hat jedoch neben dem Vorteil der hohen, detailgetreuen Auflösung den Nachteil, dass es sehr zeitintensiv ist. Die Lasergravur eines typischen Passbildes in üblicher Qualität bewegt sich beispielsweise im Bereich von zehn Sekunden.

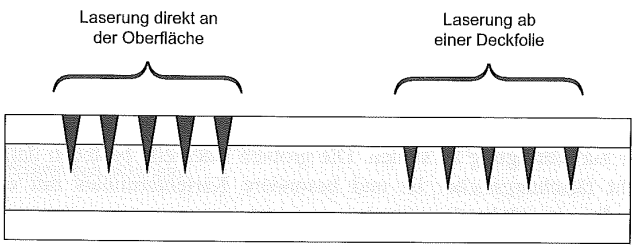


Bild 3.10 Schnittbild einer nicht maßstabsgetreu dargestellten Karte, die mit einem Laser beschriftet worden ist. Dabei gibt es zum einen die Variante der Laserung an der Kartenoberfläche und zum anderen die der Laserung unter einer für den Laser transparenten Deckfolie.

Hochprägung (*embossing*)

Eine weitere Variante des Aufbringens von Benutzerdaten auf den Kartenkörper ist die Hochprägung von Schriftzeichen. Dazu werden Metallbuchstaben mit großer Wucht auf die Kunststoffkarte geschlagen. Das Verfahren funktioniert dem Prinzip nach wie eine mechanische Schreibmaschine. Die Hochprägung hat in der heutigen Zeit nur mehr einen Grund, der aber im praktischem Einsatz sehr wichtig ist. Die Zeichen von hochgeprägten Karten lassen sich ohne großen Aufwand mit Kohlepapier auf vorgedruckte Formulare übertragen. Bei Kreditkarten ist dies beispielsweise immer noch die weltweit häufigste Art zu bezahlen.

Hochprägung lässt sich sehr leicht manipulieren. Man muss nur die hochgeprägten Buchstaben und Ziffern mäßig erhitzen (z. B. mit einem Bügeleisen), und der geprägte Kunststoff wird wieder flach. Um diese Manipulationsmöglichkeit zu verhindern, legt man oft einige Zeichen der Hochprägung überlappend in das Hologramm, denn dieses würde beim Erhitzen zerstört werden.

Thermochrome-Anzeige

Es gibt durchaus Anwendungen, bei denen es wünschenswert ist, den Kartenaufdruck in Form von Text und Bildern im Laufe der Zeit zu ändern. Ein griffiges Beispiel dazu ist ein Studentenausweis in Form einer Chipkarte, der zweimal im Jahr verlängert werden muss. Idealerweise sollte das Verfallsdatum aber ohne technische Hilfsmittel visuell lesbar sein. Dies bedeutet, dass man es auf die Karte aufdrucken muss und es nicht nur im Chip speichern kann. Ein ähnliches Beispiel sind elektronische Geldbörsen auf Chipkartenbasis, bei denen man einen Taschenkartenleser benutzen muss, um den aktuellen – und im Chip gespeicherten – Börsensaldo darzustellen.

Chipkarten mit vom Mikrocontroller angesteuertem Display sind zum jetzigen Zeitpunkt zwar technisch möglich, doch für Massenanwendungen noch zu teuer. Die Thermochrome-Anzeige (TC-Anzeige) auf der Chipkarte ist eine einfache Alternative, die zwar gegenüber einem echten Display einige Nachteile aufweist, dafür aber billig und bereits verfügbar ist. Dies ist ein zusätzliches Kartenelement, auf das von speziellen Kartenlesern Zeichen und Bilder reversibel gedruckt werden können.

Das technische Funktionsprinzip ist verhältnismäßig einfach. Ein Druckkopf mit 200 dpi oder 300 dpi Auflösung, wie er auch für Thermotransfer- und Thermosublimationsdruck verwendet wird, erhitzt an den zu schwärzenden Stellen den auf die Karte laminierten und 10 µm bis 15 µm dicken Thermochromstreifen. Dieser färbt sich dabei an den bis zu 120 °C erhitzten Stellen dunkel. Die Färbung kann durch Erhitzen des gesamten TC Streifens wieder in einen annähernd transparenten Zustand überführt werden, was dann de facto einem Löschvorgang gleichkommt.

Das Thermochrome-Verfahren ist somit zurzeit die einzige wirtschaftliche Möglichkeit, temporäre und ohne technische Hilfsmittel lesbare Daten auf der Kartenoberfläche dem Kartenbenutzer zur Verfügung zu stellen. Die größten Nachteile bestehen darin, dass das Verfahren nicht fälschungssicher ist und besondere Kartenterminals mit eingebautem Thermochrome-Druckwerk benötigt werden.

Moduliertes Merkmal (MM-Verfahren)

Im Jahr 1979 hat die deutsche Kreditwirtschaft beschlossen, alle deutschen ec-Karten mit einem maschinell lesbaren Sicherheitsmerkmal auszustatten. Nach Prüfung mehrerer unterschiedlicher Methoden wurde das von der Firma GAO entwickelte MM-Verfahren als das Sicherungsverfahren für die deutschen ec-Karten ausgewählt. Dieses Sicherheitsmerkmal wird immer noch in allen deutschen ec-Karten eingesetzt, obwohl diese mittlerweile mit einem Mikrocontrollerchip ausgerüstet sind. Der Zweck dieses Merkmals besteht darin, das unbefugte Kopieren und Manipulieren des Magnetstreifens zu verhindern.

Das MM-Verfahren ist ein typisches Beispiel für ein geheimes und sehr effektives Sicherheitsmerkmal. Es wird seit über zwei Jahrzehnten in Millionen von Karten eingesetzt. Ein Artikel von Siegfried Otto [Otto 82] beschreibt überblickshaft den grundsätzlichen Aufbau dieses Sicherheitsmerkmals.

Der Begriff „MM-Verfahren" leitet sich von dem Terminus „moduliertes Merkmal" ab und kann als ein Hinweis für eine im Innern des Kartenkörpers eingebrachte, maschinell lesbare Substanz aufgefasst werden [Meyer 96]. Dieser MM-Code wird bei der Kartenprüfung von einem besonderen Sensor aus der Karte ausgelesen und an ein Sicherheitsmodul, die MM-Box, weitergegeben. Diese MM-Box erhält zusätzlich den gesamten Dateninhalt

des Magnetstreifens und insbesondere den MM-Prüfwert, welcher ebenfalls auf dem Magnetstreifen gespeichert ist. Im Innern der MM-Box wird nun eine auf dem DES basierende Einwegfunktion angestoßen, die aus Magnetstreifendaten und MM-Code einen Wert berechnet. Ist dieses Ergebnis nun identisch mit dem MM-Prüfwert, wird entschieden, dass der Magnetstreifen und die Karte zusammengehören.

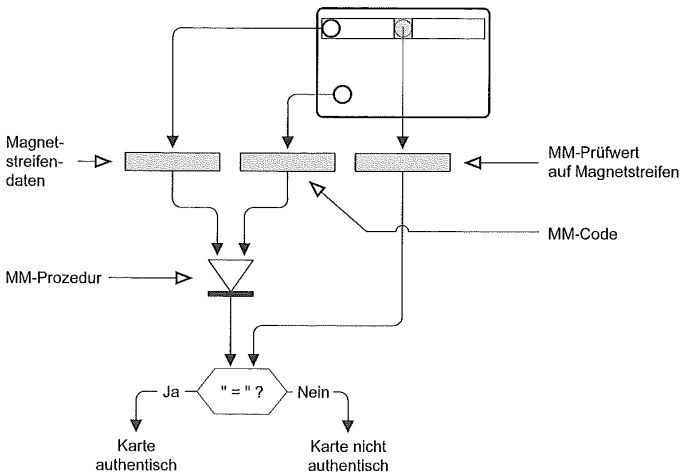


Bild 3.11 Prinzipieller Ablauf einer Echtheitsprüfung der deutschen ec-Karte bei Verwendung des MM-Verfahrens. Das Sicherheitsmodul „MM-Box" schützt die MM-Prozedur und den anschließenden Vergleich, sodass lediglich eine Ja/Nein-Aussage als Ergebnis dem übergeordneten System gemeldet wird.

Würde man nun versuchen, auf dem Magnetstreifen einer Blankokarte einen gültigen Magnetstreifeninhalt zu schreiben, so würde dies erkannt, da eine Blankokarte kein MM-Merkmal besitzt. Ebenso würde es erkannt, wenn man den Magnetstreifen einer ec-Karte auf eine andere ec-Karte kopiert, da dann der MM-Prüfwert falsch wäre. Das MM-Merkmal ist unsichtbar und seine Position in der Karte und die genaue Funktionsweise sind unbekannt. Es wird des Weiteren aus Materialien und mithilfe einer Technologie hergestellt, die auf dem Markt nicht käuflich erhältlich sind.

Die MM-Box ist in allen deutschen Geldausgabeautomaten und auch in manchen POS-Terminals eingebaut. Diese Geräte erlangen dadurch die Fähigkeit, die Zusammengehörigkeit von Magnetstreifen und Kartenkörper zu prüfen. Das Verfahren als solches ist durch keine Norm festgelegt und wird nur in Deutschland benutzt. Durch das MM-Verfahren sind die Magnetstreifen der deutschen ec-Karten vor dem technisch mittlerweile völlig problemlosen Kopieren des Magnetstreifens geschützt.

Da die Sicherheit des MM-Verfahrens nicht ausschließlich auf einigen geheimen Schlüsseln beruht, sondern vor allem auf Materialien und Technologie, kann es nur durch Geheimhaltung gegen Angriffe geschützt werden. Im Übrigen ist Vertraulichkeit bei dieser Art von physikalischen Sicherheitsmerkmalen ein weithin übliches und auch durchaus notwendiges Mittel, um zusätzlichen Schutz zu gewährleisten. In diesen Fällen genügt es nicht, analog Kerckhoffs Prinzip, nur einige Schlüssel geheim zu halten.

Sicherheitsmerkmale (*security features*)
Gerade in dem Zeitraum zwischen Masseneinsatz von Karten ohne Chip und der Einführung von Chipkarten wurden die unterschiedlichsten visuellen Sicherheitsmerkmale entwickelt. Es war während dieser Zeit die einzige Möglichkeit, Karten auf Echtheit zu prüfen. Durch den in neuen Karten implantierten Mikrocontroller und die dadurch möglichen kryptografischen Verfahren sind diese Echtheitsmerkmale etwas in den Hintergrund getreten. Doch sind sie auch heute noch von großer Bedeutung, wenn Karten nicht durch eine Maschine, sondern durch einen Menschen auf ihre Echtheit hin überprüft werden müssen, da dieser nie ohne Hilfsmittel direkt auf den Chip zugreifen kann.

Hier konnten nur stark zusammengefasst die wesentlichen und bekanntesten Sicherheitsmerkmale von Karten aufgezählt werden. Es gibt noch eine große Zahl weiterer Merkmale, wie unsichtbare, nur im IR-/UV-Licht lesbare und magnetisch lesbare Codes oder spezielle Drucke in Regenbogenfarben. Leider lassen sich diese technisch sehr interessanten Merkmale hier aus Platzgründen nicht alle erläutern.

In Zukunft wird es nicht nur auf der Karte Sicherheitsmerkmale geben, sondern auch im Chip. Es ist denkbar, dass man mit Sicherheitschips ähnlich wie mit Banknotenpapier verfährt. Echtes Banknotenpapier mit spezifischen Echtheitsmerkmalen ist eine wesentliche Voraussetzung, um echte Geldscheine drucken zu können. Um ähnliche Merkmale im Chipbereich einzuführen, sind spezielle Chips mit modifizierter Hardware notwendig. Das Terminal kann dann die Veränderungen, d. h. das Merkmal, an der Hardware messen und aufgrund des Messergebnisses die Echtheit des Chips beurteilen.

Als Beispiel für ein Hardwaremerkmal könnte man sich vorstellen, dass ein schneller kryptografischer Algorithmus als Zusatzhardware auf einem bestimmten Chip realisiert ist. Die zur Berechnung eines bestimmten Wertes notwendige Zeit wäre durch die Hardwarelösung so klein, dass man eine Emulation mit Software auf einem anderen Chip in ähnlich kurzer Zeit nicht durchführen könnte. Damit könnte ein Terminal einen Chip durch bloße Zeitmessung von anderen unterscheiden. Es existieren Chips, die ein Hardwaremerkmal in der beschriebenen oder einer ähnlichen Form haben und dann natürlich, ähnlich wie dem Banknotenpapier, nicht mehr frei erhältlich sind. Allerdings sind solche Merkmale durch die hohen Kosten für die eigens zu entwickelnde Hardware nur für sehr große Anwendungen geeignet. Auch ist die dann fast zwangsläufig daraus folgende Monopolstellung eines bestimmten Chips ohne die Möglichkeit eines Zweitlieferanten für viele Kartenherausgeber schwierig zu akzeptieren. Doch Hardwaresicherheit ist eine wichtige Komponente in der Sicherheitsarchitektur eines Chipkartensystems, und sie ist leider nicht zum Nulltarif zu haben.

3.2 Kartenkörper

Die Materialien, der Aufbau und die Herstellung des Kartenkörpers werden im Wesentlichen durch die Funktionselemente der Karte sowie durch die Belastung der Karte bei der Handhabung während der Anwendung bestimmt. Typische Funktions- oder Kartenelemente der Karte sind:

- Magnetpiste
- Unterschriftsstreifen
- Hochprägung

- Darstellung von Personalisierungsdaten durch Laserstrahl (Schrift, Foto, Fingerabdruck)
- Hologramm
- Sicherheitsdruck
- Unsichtbare Echtheitsmerkmale (z. B. Fluoreszenz)
- Chip mit Kontakten oder anderen Koppelelementen

Man sieht, die relativ kleine Karte mit ihrer Dicke von nur 0,76 mm muss unter Umständen eine Vielzahl von Funktionselementen aufnehmen. Dies stellt höchste Anforderungen an die Qualität der Materialien sowie des Herstellungsprozesses.

Die Mindestanforderungen an die Belastbarkeit der Karte sind in den ISO-Normen 7810, 7813 und 7816 Teil 1 festgelegt. Diese Anforderungen betreffen im Wesentlichen folgende Themen:

- UV-Strahlung
- Röntgen-Strahlung
- Oberflächenprofil der Karte
- Mechanische Belastbarkeit der Karte und der Kontakte
- Elektromagnetische Verträglichkeit
- Elektrostatische Entladungen
- Temperaturbeständigkeit

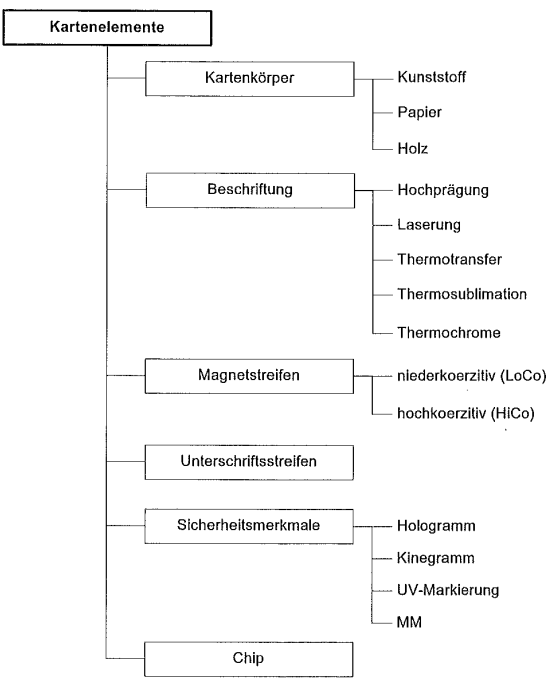


Bild 3.12 Die Systematik der Kartenelemente.

Im ISO-Standard ISO/IEC 10373 sind für viele dieser Forderungen Testmethoden festgelegt, die dem Anwender und Hersteller von Karten eine objektive Prüfung der Kartenqualität ermöglichen. Für Chipkarten sind dabei die Biege- und Torsionstests von besonderer Wichtigkeit. Der Chip ist nämlich wegen seiner Materialeigenschaften (spröde und zerbrechlich wie Glas) ein empfindlicher Fremdkörper in der elastischen Karte und muss durch besondere konstruktive Maßnahmen gegen mechanische Beanspruchung bei Biegung und Torsion der Karte geschützt werden. In Kapitel 9 befindet sich eine detaillierte Aufstellung von Tests und dazu angewandter Methoden.

3.2.1 Kartenmaterialien

Das als Erstes und auch heute noch am meisten verwendete Material für Identifikationskarten ist der amorphe Thermoplast PVC (Polyvinylchlorid). Es ist das preisgünstigste aller verfügbaren Materialien, lässt sich gut verarbeiten und deckt ein weites Einsatzspektrum ab. So kommt es beispielsweise weltweit bei Kreditkarten zum Einsatz. Nachteile von PVC sind die begrenzte Lebensdauer wegen der Alterung des Materials sowie die geringe Temperaturbeständigkeit. Als Ausgangsmaterial für die Kartenkörperproduktion wird PVC als Folie verwendet, der Spritzguss von PVC ist nicht möglich. Die weltweite Produktion an PVC betrug im Jahr 1996 ungefähr 13 Millionen Tonnen, davon entfielen 35000 Tonnen auf Karten, was einem Anteil von ca. 0,27 % entspricht. PVC gilt allgemein als umweltschädlich. So ist der Ausgangsstoff Vinylchlorid als krebserregend eingestuft, und bei der Verbrennung kann Salzsäure und unter ungünstigen Bedingungen sogar Dioxin entstehen. Weiterhin enthält PVC zur Stabilisierung oft Schwermetalle. Allerdings ist PVC noch immer mit Abstand der am häufigsten benutzte Kunststoff für Karten. Dies ist vor allem durch den Preis und die gute Verarbeitbarkeit begründet. Aufgrund der schlechten Umwelteigenschaften sinkt aber der Einsatz von PVC von Jahr zu Jahr. Viele Kartenherausgeber lehnen mittlerweile dieses Material aus umweltpolitischen Erwägungen prinzipiell ab.

Um die Nachteile von PVC zu umgehen, werden seit einiger Zeit Karten aus ABS (Acrylnitril-Butadien-Styrol) gefertigt. Dieses Material ist ebenfalls ein amorpher Thermoplast und zeichnet sich durch hohe Festigkeit und Temperaturbeständigkeit aus. Deshalb ist ABS das üblicherweise bei Mobilfunkkarten eingesetzte Material, die aus naheliegenden

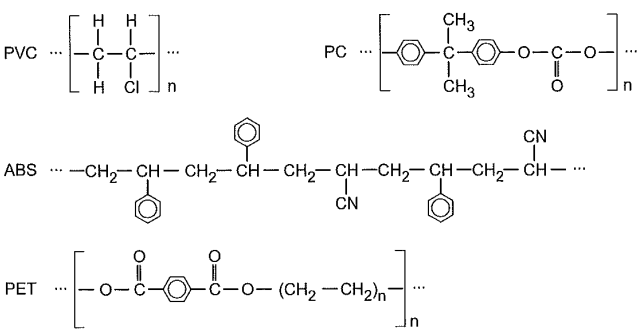


Bild 3.13 Strukturformeln der wichtigsten Kunststoffe für Kartenkörper.

genden Gründen sehr hohen Temperaturbelastungen ausgesetzt sind. ABS lässt sich sowohl als Folie als auch im Spritzguss gut verarbeiten. Die Hauptnachteile dieses Materials sind seine eingeschränkte Farbgebung sowie die Empfindlichkeit gegen Witterungseinflüsse. Der Ausgangsstoff Benzol ist krebserregend, doch sonst sind keine die Umwelt betreffenden Nachteile bekannt.

Für Anwendungen, bei denen hohe Festigkeit und Langlebigkeit gefordert sind, wird Polycarbonat (PC) eingesetzt. Es ist ein typisches Material für Ausweiskarten und, nebenbei bemerkt, auch das Trägermaterial für CDs und DVDs. Aufgrund der guten Wärmebeständigkeit von Polycarbonat benötigt man zum Aufbringen von Hologrammen oder Magnetstreifen im Hot-Stamp-Verfahren relativ hohe Temperaturen, was leicht zu Problemen mit der Temperaturbeständigkeit des aufzubringenden Materials führen kann. Die Hauptnachteile von Polycarbonat sind die Kerbempfindlichkeit sowie der gegenüber allen ande-

Tabelle 3.1 Die üblichen Materialien für Kartenkörper und ihre Eigenschaften im Überblick.[1] Die Preise sind relativ zu PVC angegeben.

Eigenschaften	**PVC**	**ABS**	**PC**	**PET**
Haupteinsatz	Kreditkarten	Mobiltelefonkarten	ID-Karten	KV-Karten
Haupteigenschaft	kostengünstig	temperaturbeständig	langlebig	umweltfreundlich
Kartenherstellung	nur Folie	Folie und Spritzguss	Folie und Spritzguss	Folie und Spritzguss
Wärmebeständigkeit	65–90 °C	75–100 °C	160 °C	bis 80 °C
Kältebeständigkeit	mittel	hoch	mittel	mittel
mechanische Festigkeit	gut	gut	gut	sehr gut
Embossing	gut	schlecht	gut	gut
Bedrucken	gut	mittel	mittel	gut
Hot-Stamping (z. B.: für Hologramme u. Ä.)	gut	gut	schwierig	gut
Lasergravur	ja	schlecht	gut	gut
typ. Lebensdauer	ca. 2 Jahre	ca. 3 Jahre	ca. 5 Jahre	ca. 3 Jahre
Anteil an der weltweiten Kartenproduktion (1998)	85 %	8 %	5 %	2 %
Preis	1 fach	2 fach	7 fach	2,5 fach
Umwelt	• u. U. Bildung von Dioxin bei Verbrennung • Stabilisatoren aus Schwermetallen	• krebserregendes Benzol ist einer der Ausgangsstoffe • bei der Verbrennung kann Blausäure entstehen	• Phosgen/Chlor sind für die Herstellung notwendig • verbrennt schadstoffarm	• z.Zt. umweltfreundlichstes Kartenmaterial • verbrennt schadstoffarm
Besonderheiten	negatives Image		kerbempfindlich	

[1] z. T. nach [Houdeau 97, Grün 96]

ren Kartenmaterialien sehr hohe Preis. Ungünstigerweise sind für die Herstellung von Polycarbonat die beiden umwelttechnisch bedenklichen Stoffe Phosgen und Chlor erforderlich. Polycarbonatkarten sind sehr leicht am „blechernen" Klang zu erkennen, wenn man sie auf einen harten Untergrund fallen lässt.

Ein umweltfreundliches Ersatzmaterial vor allem für PVC, das im Verpackungsbereich schon seit längerer Zeit eingesetzt wird, ist Polyethylenterephtalat (PET), dessen umgangssprachlicher Name Polyester lautet. Dieser thermoplastische Kunststoff wird bei Chipkarten sowohl in seiner amorphen (A-PET) als auch in der teilkristallinen (PETP) Form verwendet. Beide eignen sich sowohl für Folien- als auch für Spritzgussverarbeitung. Allerdings lässt sich das teilkristalline PET schwierig laminieren, was zusätzliche Arbeitsschritte bei der Fertigung notwendig macht.

Es gibt immer wieder Versuche, neben den vier üblichen Materialien für Kartenkörper PVC, ABS, PC und PET verbesserte oder neue Kunststoffe zu verwenden. Ein Beispiel dafür ist Zelluloseacetat, das zwar gute Umwelteigenschaften hat, sich aber bislang als sehr schwierig für die Massenfertigung von Karten herausstellte. Echte alternative Materialien zu Kartenkörpern aus Kunststoffen, wie etwa Papier, sind oft diskutiert, doch bisher nirgends in relevanten Stückzahlen eingesetzt worden. Die Anforderungen an Preis, Lebensdauer und Qualität an eine Karte sind nun einmal sehr hoch und momentan nur von Kunststoffen erfüllbar.

Keine echte Alternative zu den Kunststoffkartenkörpern, aber zumindest ein interessant (oder auch kurios) zu nennender Feldversuch wurde 1996/97 in Dänemark von Danmønt durchgeführt.[2] Man gab etwa 6 000 Chipkarten mit einem Kartenkörper aus schichtlaminiertem und bedrucktem Birkenholz aus (8 Schichten zu je 0,1 mm). Die emittierten Karten bestanden zwar nicht die diversen Tests nach ISO 10 373, wie etwa Biegung und Torsion, und waren logischerweise auch nicht hochprägbar, doch etwa 90 % der Benutzer äußerten sich positiv und ließen erkennen, dass sie keine Probleme mit ihren Karten hatten. Aus Umweltsicht ist die Chipkarte aus Birkenholz leider keine Innovation, da die Holzschichten mit einem Kunststoffklebstoff zusammenlaminiert werden müssen und auch die üblichen Druckfarben erforderlich sind.

3.2.2 Chipmodule

Das wichtigste Kartenelement einer Chipkarte ist natürlich der Chip. Dieses sehr empfindliche Bauteil kann allerdings nicht einfach wie ein Magnetstreifen auf die Kartenoberfläche laminiert werden, sondern benötigt zum Schutz vor dem rauen Kartenalltag eine Art Gehäuse – das Modul. Zusätzlich zur Sicherung vor Umwelteinflüssen benötigen die kontaktbehafteten Chipkarten noch sechs bzw. acht Kontaktflächen, um Energieversorgung und Datenübertragung mit dem Terminal aufbauen zu können. Ein Teil der Moduloberfläche dient dieser Funktion der galvanischen Kontaktierung mit der äußeren Welt. Das Chipmodul sollte natürlich möglichst kostengünstig sein.

Um die beiden technischen Funktionen, Schutz des zerbrechlichen Halbleiterchips und Verfügungsstellung von Kontaktflächen, zu bewerkstelligen, wurden seit Anfang der Chipkartenentwicklung die unterschiedlichsten Modulbauformen ersonnen. Die wichtigsten davon werden im Folgenden detailliert dargestellt.

---

[2] [a la Card 97]

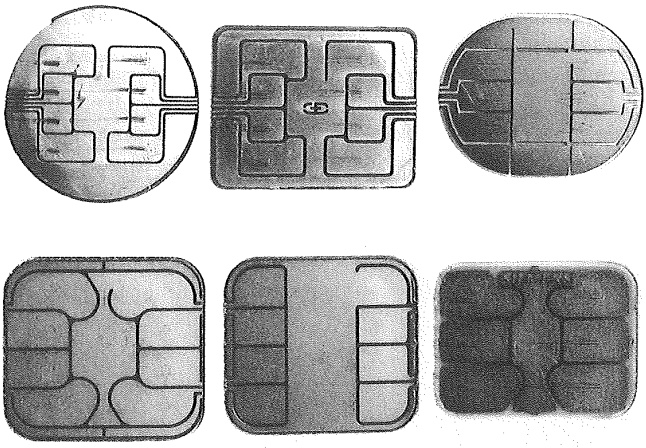


Bild 3.14 Die Evolution von Chipmodulen der Chip-on-Flex-Technik anhand einiger Beispiele, beginnend oben links mit einigen der ersten Chip-on-Flex-Module mit 8 Kontakten bis zu den heutigen Modulen mit 8 bzw. 6 Kontakten.

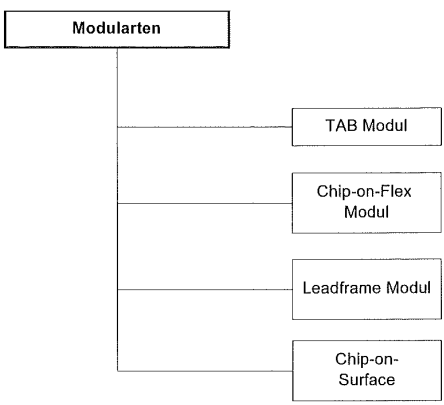


Bild 3.15 Klassifizierungsbaum der unterschiedlichen Modularten.

3.2.2.1 Elektrische Kontaktierung zwischen Chip und Modul

Zwischen dem Chip im Modul und den an der Modulaußenseite befindlichen Kontaktelementen muss eine elektrische Verbindung hergestellt werden. Dafür gibt es heute prinzipiell zwei gängige Verfahren: Beim Draht-Bond-Verfahren (Wire-Bonding) wird mit einem automatischen Bonder ein nur wenige Mikrometer dünner Golddraht zwischen Chip und der Rückseite der Kontaktflächen gelegt. Die elektrische Verbindung zwischen Draht und Chip bzw. Modul wird durch Ultraschallschweißen hergestellt. Die Chipoberfläche ist bei diesem Verfahren immer in entgegengesetzter Richtung der Kontaktflächen orientiert. Dieses Verfahren ist in der Halbleiterindustrie seit langer Zeit Standard und lässt sich auch problemlos für die Massenproduktion von Chipmodulen einsetzen. Allerdings muss jeder

Chip mit fünf Drähten elektrisch mit dem Modul verbunden werden, was durchaus Zeit und somit Geld kostet.

Eine Weiterentwicklung zur Reduzierung der Kosten des Chipeinbaus in Module ist das Die-Bonding. Bei diesem Verfahren findet die elektrischen Verbindung zwischen Chip und Modul nicht durch Drähte statt, sondern der Chip wird mit der Modulrückseite so mechanisch verbunden, dass jeweils ein Kontakt für die fünf elektrischen Verbindungen hergestellt ist.

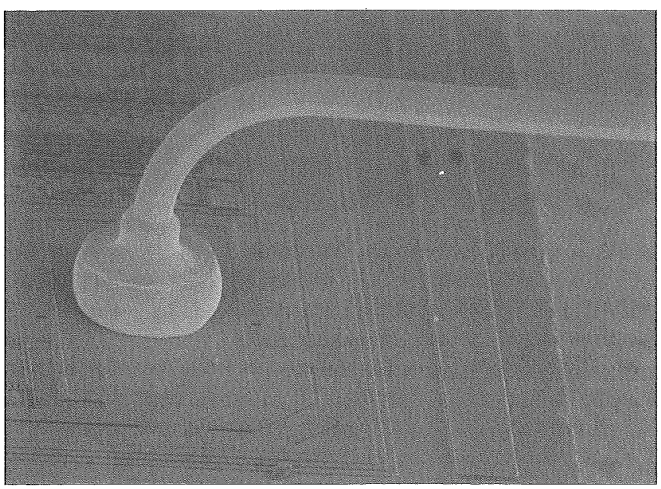


Bild 3.16 Verbindungsstelle zwischen Bonddraht und Bondpad auf einem Chipkarten-Mikrocontroller in 1 000facher Vergrößerung. (Giesecke & Devrient)

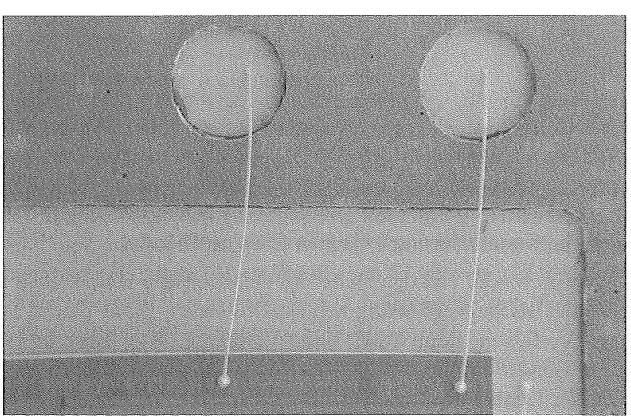


Bild 3.17 Aufsicht der elektrischen Verbindung zwischen Chipkarten-Mikrocontroller (am unteren Bildrand) und dem Chipkarten-Modul (am oberen Bildrand) in 400facher Vergrößerung. (Giesecke & Devrient)

3.2.2.2 TAB-Modul

Die TAB-Technik (*tape automated bonding*) wird mittlerweile selten eingesetzt, obwohl sie noch Anfang der Neunzigerjahre ein Standardverfahren für Chipverpackung in großen Stückzahlen war. Es ist mittlerweile jedoch technisch überholt und auch zu teuer. Wir führen es hier vor allem der Vollständigkeit halber auf.

Ein Chipmodul in TAB-Technik ist in Bild 3.18 dargestellt. Das Besondere an dieser Technik besteht darin, dass zunächst auf die Anschlussflächen (pads) des Chips metallische Höcker (*bumps*) galvanisch aufgebracht werden. Auf diese Höcker werden dann die Leiterbahnen des Trägerfilms aufgelötet. Diese Lötverbindung ist mechanisch so belastbar, dass der Chip selbst nicht weiter befestigt werden muss, sondern nur an den Leiterbahnen hängt. Die aktive Fläche des Chips wird noch durch eine Abdeckmasse gegen Umwelteinflüsse geschützt. Der Vorteil des TAB-Verfahrens liegt in der hohen mechanischen Belastbarkeit der Chipanschlüsse und in der geringen Bauhöhe des Moduls. Dieser Vorteil muss allerdings durch einen höheren Preis gegenüber den anderen Modul-Techniken erkauft werden.

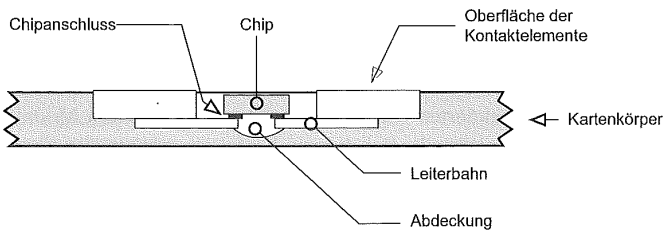


Bild 3.18 Querschnitt durch ein Chipmodul in der TAB-Technik.

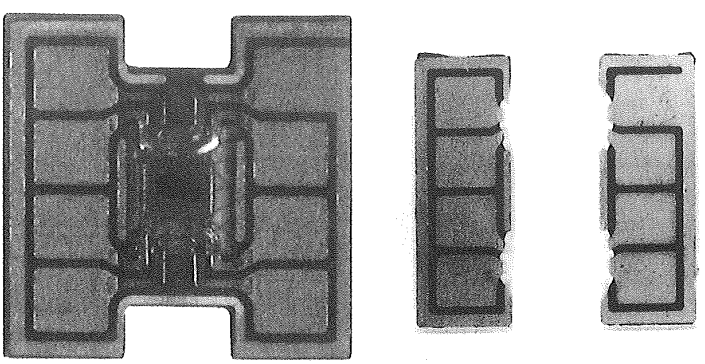


Bild 3.19 Auf der linken Seite ist ein TAB-Modul im uneingebauten Zustand zu sehen und auf der rechten Seite ein in eine Chipkarte eingebautes TAB-Modul.

Der Einbau eines TAB-Moduls in eine Karte ist nicht einfach, da er schon bei der Laminierung der einzelnen Folien berücksichtigt werden muss. Vor dem Laminieren werden passende Aussparungen in die Folien gestanzt und passend dazu die Chipmodule eingesetzt. Beim Laminiervorgang wird dann das Chipmodul fest mit dem Kartenkörper ver-

schweißt. Mit diesem Verfahren lassen sich hohe Ansprüche an die Verbundqualität zwischen Chipmodul und Kartenkörper erfüllen. Der Chip kann praktisch nicht mehr aus der Karte herausgelöst werden, ohne diese zu zerstören.

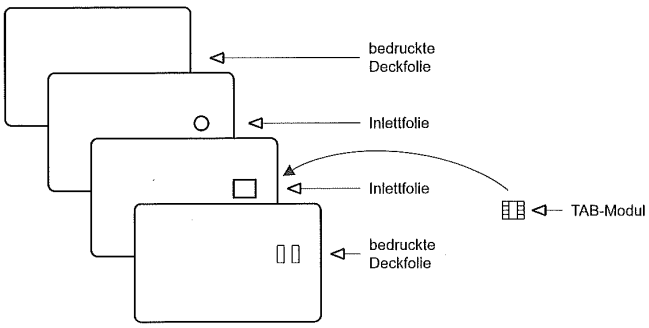


Bild 3.20 Einbringen des TAB-Moduls beim Laminierprozess.

3.2.2.3 Chip-on-Flex-Modul

Am weitesten verbreitet sind heute Module in der Chip-on-Flex-Technik und mit im Draht-Bond-Verfahren kontaktierten Chips. Der Aufbau eines solchen Moduls ist in Bild 3.22 als Schnittzeichnung dargestellt. Bei diesem Verfahren wird in den fertigen Kartenkörper ein Loch gefräst, in das dann die Chipmodule eingeklebt werden.

Das Trägermaterial ist eine dünne Leiterplatte aus glasfaserverstärktem Epoxidharz mit einer Dicke von 120 µm. Die späteren Kontaktflächen bestehen aus 35 bzw. 75 µm auflaminiertem Kupfer, das in einem folgenden Prozessschritt galvanisch vergoldet wird. Damit wird die Kontaktfläche unempfindlich gegenüber die elektrische Leitfähigkeit einschrän-

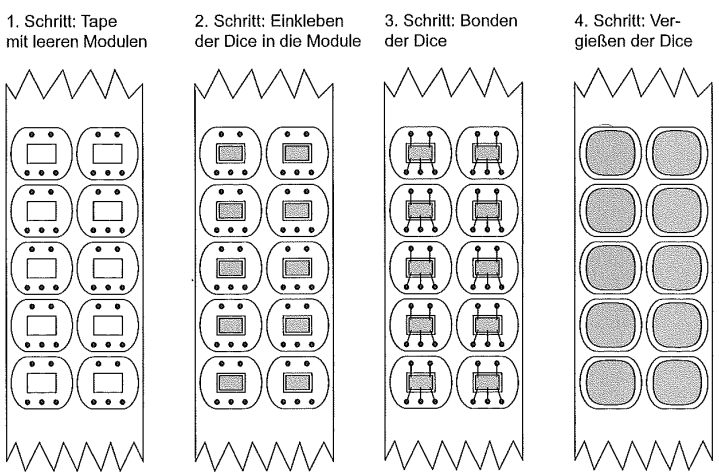


Bild 3.21 Darstellung der vier wesentlichen Prozessschritte bei der Herstellung eines Chip-on-Flex-Moduls.

kenden Einflüssen wie beispielsweise Oxidation. Zur Aufnahme des Chips und der Drahtverbindungen sind Löcher aus dem Träger ausgestanzt. Der ca. 200 µm dicke Chip wird mit einem Pick-and-Place-Automat vom gesägten Wafer genommen und auf der Rückseite des Moduls in die vorgesehene Ausstanzung der Leiterplatte befestigt. Anschließend werden die Chipanschlüsse mit einigen Mikrometer dünnen Bonddrähten mit der Rückseite der Kontaktflächen verbunden. Zum Schluss werden der Chip und die Bond-Drähte durch eine Vergussmasse gegen Umwelteinflüsse geschützt. Die Gesamtdicke des fertigen Moduls bewegt sich dann typischerweise in einem Bereich von 600 µm.

Der Vorteil dieses Verfahrens liegt darin, dass es sich weitgehend an das in der Halbleiterindustrie übliche Verfahren zur Verpackung von Chips in Standardgehäusen anlehnt. Es ist weniger spezielles Know-how erforderlich als beim TAB-Verfahren, was sich günstig auf den Preis auswirkt. Dieses Verfahren ist auch gut einsetzbar, wenn sehr komplexe Kartenkörper mit vielen Funktionselementen produziert werden sollen. Karten mit Fertigungsfehlern lassen sich dann nämlich aussortieren, bevor die teuren Chips eingebaut werden. Der Nachteil liegt darin, dass sowohl die Bauhöhe wie auch die Länge und Breite des Moduls deutlich größer sind als beim TAB-Modul, weil nicht nur der Chip, sondern auch die Bond-Drähte durch die Abdeckmasse geschützt werden müssen. Dies ist vor allem deshalb ungünstig, weil die genormte Chipkartendicke von 0,76 mm nur sehr wenig Spielraum für allzu dicke Module lässt.

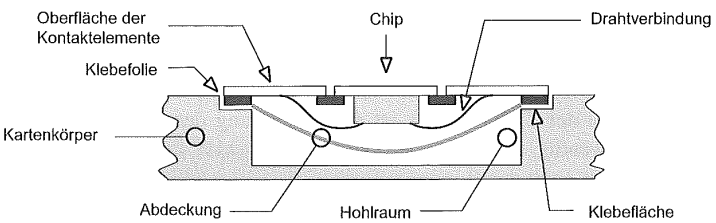


Bild 3.22 Querschnitt durch ein Chipmodul in der Chip-on-Flex-Technik.

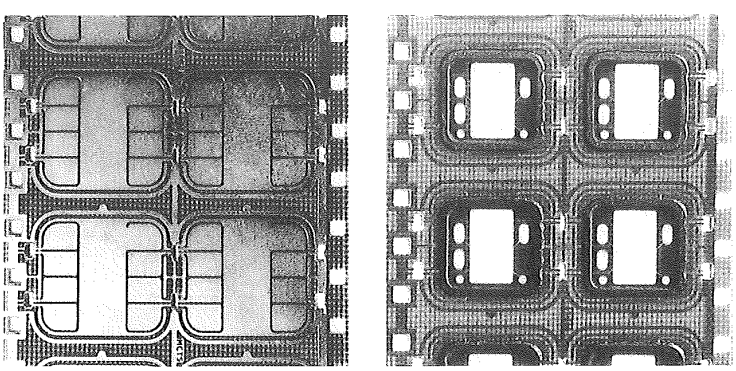


Bild 3.23 Vorder- und Rückseite von Modulen in Chip-On-Flex-Technik auf 35 mm Tape. Man sieht an der Rückseite deutlich die fünf Aussparungen auf der Trägerplatine, für die elektrische Verbindung zum Chip durch Bonddrähte.

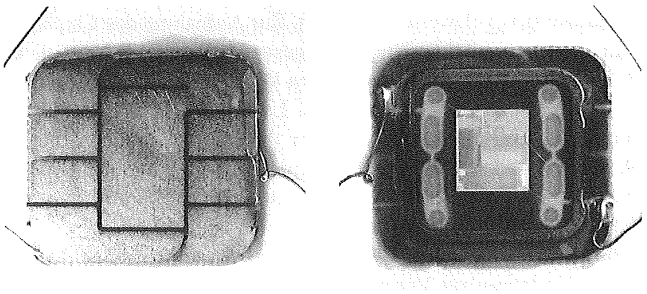


Bild 3.24 Vorder- und Rückseite eines Moduls in Chip-On-Flex-Technik für eine Dual-Interface-Karte.

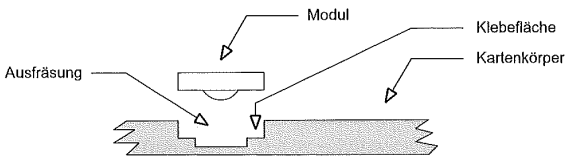


Bild 3.25 Einbringen des Chipmoduls in einen gefrästen Kartenkörper.

3.2.2.4 Lead-Frame-Modul

Technisch gesehen ist sowohl TAB-Technik als auch die Chip-on-Flex-Technik nicht optimal, da beide Verfahren wenig Raum für eine Reduktion des fertigungstechnischen Aufwands bieten. Bei der TAB-Technik ist die Herstellung des Kartenkörpers aufgrund des Moduls sehr aufwendig, und bei der Chip-on-Flex-Technik sorgen sowohl der komplizierte Modulkörper als auch das Bonden für ungünstige Herstellkosten. Diese Gründe führten zur Entwicklung eines Modultyps, der mechanisch genauso stabil wie TAB- und Chip-on-Flex-Technik ist, aber kostengünstiger zu produzieren: das Lead-Frame-Modul.

Der Aufbau eines Lead-Frame-Moduls ist relativ unkompliziert. Die aus einer Kupferlegierung mit galvanisierter Goldoberfläche gestanzten Kontaktflächen werden durch einen so genannten Moldkörper aus Kunststoff zusammengehalten. Auf diesem wird dann mit Pick-and-Place der Chip aufgesetzt und im Wire-Bond-Verfahren mit den Rückseiten der Kontaktflächen verbunden. Anschließend schützt man den Chip noch mit einem Überzug aus lichtundurchlässigem, meist schwarzem Epoxidharz.

Das Lead-Frame-Verfahren ist zurzeit eines der billigsten Verfahren zur Modulherstellung, ohne Nachteile bei der mechanischen Stabilität dafür in Kauf nehmen zu müssen.

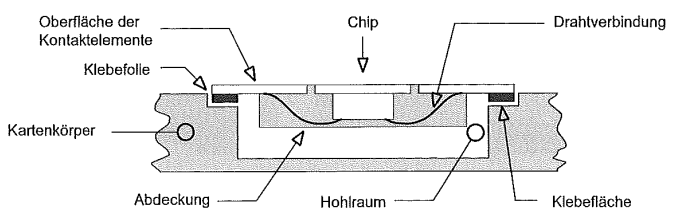


Bild 3.26 Querschnitt durch ein Chipmodul in der Lead-Frame-Technik.

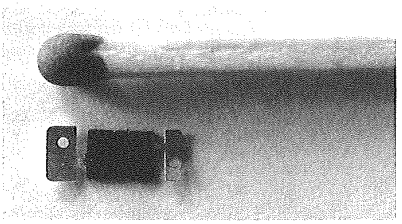


Bild 3.27 Ausgestanztes Modul in Lead-Frame-Technik mit den beiden Spulenanschlüssen für kontaktlose Chipkarten im Vergleich mit einem Streichholz.

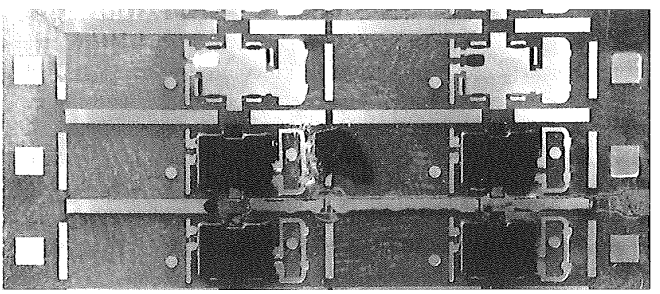


Bild 3.28 Paarweise nebeneinander angeordnete Lead-Frame-Module auf einem 35 mm Tape für kontaktlose Chipkarten. Am oberen Bildrand sind die beiden leeren Plätze von bereits ausgestanzten Modulen zu erkennen.

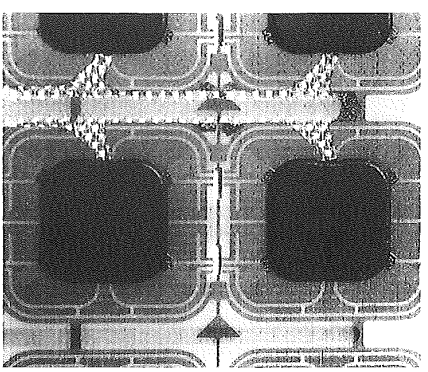


Bild 3.29 Paarweise nebeneinander angeordnete Lead-Frame-Module auf einem 35 mm Tape für kontaktbehaftete Chipkarten.

3.2.2.5 Chip-On-Surface-Verfahren

Für flächenmäßig kleine Chips existiert seit Mitte der Neunzigerjahre neben den bekannten Verfahren des Chipeinbaus in ein Modul eine technisch sehr interessante Alternative. Die von Solaic [Sligos] entwickelte MOSAIC (Microchip On Surface And In Card) -Technologie benötigt kein Modul für den Chip, denn dieser befindet sich direkt auf dem Kartenkörper.

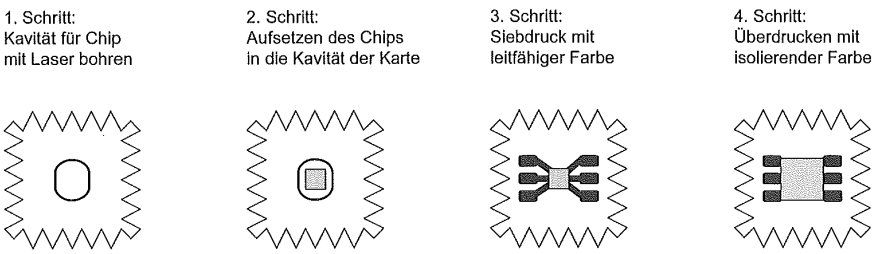


Bild 3.30 Die vier Fertigungsschritte beim Chip-on-Surface-Verfahren zur Herstellung einer Karte mit Chip.

Das MOSAIC-Verfahren eignet sich für Chipgrößen von etwa 1 mm^2, was den Einsatz zurzeit auf reine Speicherchips reduziert, da Mikrocontroller für diese Technologie noch zu groß sind. Die Technik funktioniert folgendermaßen: Zuerst wird mit einem Laser an der späteren Chip-Position Material abgetragen und dann an diese Stelle der Speicherchip geklebt. Im nächsten Arbeitsschritt werden sowohl der Chip mit einer Silberleitpaste im Siebdruckverfahren an seinen Anschlussstellen überdruckt als auch gleichzeitig die Kontaktflächen auf die Karte aufgedruckt. Im abschließenden Arbeitsschritt wird der Chip und die gedruckten Leiterbahnen zu den Kontaktflächen mit einem nicht leitendem Lack abgedeckt. Damit sind sie einerseits nach außen hin elektrisch isoliert und andererseits gegenüber Umwelteinflüssen abgesichert.

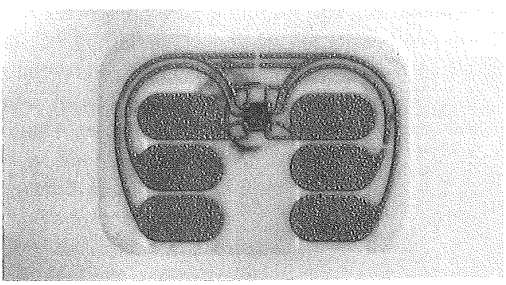


Bild 3.31 Speicherchip mit einer Seitenlänge von 0,5 mm (= 0,25 mm^2) mit Kontaktflächen nach ISO/IEC 7816–3, der samt den Kontakten im Chip-On-Surface-Verfahren auf eine Telefonkarte aufgebracht wurde.

Die Darstellung zeigt deutlich, dass das Verfahren sehr gut für hohe Stückzahlen in der Massenproduktion von Karten geeignet ist, da es im Wesentlichen nur aus einem kurzen Laserbeschuss des Kartenkörpers und zwei Druckvorgängen besteht. Diese Technologie erfordert aber ein äußerst positionsgenaues Druckverfahren, damit die Anschlussstellen am Chip genau getroffen werden. Als Kartenmaterial wurde bislang hauptsächlich Polycarbonat verwendet, das sich für das Chip-On-Surface-Verfahren besonders gut eignet. Der Durchsatz an produzierten Chipkarten bewegt sich im Bereich von 5000 Stück pro Stunde und Maschine.

Bei Modulen in Flip-Chip-Bauweise wird der Chip mit seiner Vorderseite auf die Rückseite des Moduls aufgebraucht und elektrisch verbunden. Anschließend wird das so zusammengefügte Modul vergossen. Man bezeichnet diese kostengünstigen modernen Module üblicherweise als FCOS (*flip chip on substrate*).

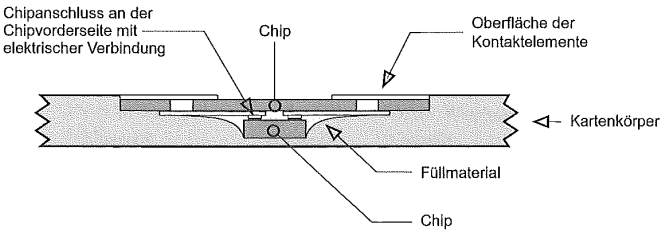


Bild 3.32 Querschnitt durch ein Chipmodul in der Flip-Chip-Technik.

3.3 Elektrische Eigenschaften

Die elektrischen Eigenschaften von Chipkarten sind nur vom implantierten Mikrocontroller abhängig, da er das einzige elektrisch beschaltete Bauelement auf den Chipkarten darstellt. Ausnahmen werden hier in Zukunft sicherlich Chipkarten mit zusätzlichen Kartenelementen wie Display, Tastatur und Ähnlichem sein, doch wird bis zum Masseneinsatz dieses neuen Typus von Chipkarten noch einige Zeit vergehen.

Die Anwendung, die von Anfang an bei den elektrischen Eigenschaften viele rigide Vorgaben setzte, waren die Chipkarten für das Mobilfunksystem GSM. Dieses System, bei dem technisch unterschiedlichste Terminaltypen, die auch noch von den unterschiedlichsten Herstellern stammen, mit einer mindestens ebenso großen Anzahl von Kartentypen zusammenarbeiten müssen, setzt seit langem die engsten Rahmenbedingungen. Bedingt durch die großen Stückzahlen an eingesetzten Chipkarten haben die bei GSM festgelegten elektrischen Eigenschaften Richtcharakter für alle Hersteller von Chipkarten-Mikrocontrollern. Man kann davon ausgehen, dass beinahe alle Neuentwicklungen von Mikrocontrollern für Chipkarten die elektrischen Rahmenwerte der entsprechenden GSM-Spezifikationen erfüllen, da sie sonst zumindest im Telekommunikationsbereich nicht zu verkaufen sind.

In den Anfängen der Chipkartentechnik war oft in erster Linie entscheidend, dass der implantierte Mikrocontroller funktionsfähig war, und weniger seine elektrischen Eigenschaften wie beispielsweise sein Stromverbrauch. Es gab damals fast ausnahmslos geschlossene Anwendungen die einen einzigen Typ von Chipkarten und einen dazu passend entwickelten Terminaltyp verwendeten. Die elektrischen Eigenschaften der Chipkarte waren nur insoweit relevant, als sie konstant zu sein hatten, da man die Terminals auf den verwendeten Mikrocontroller auslegte. Dies hat sich aber in der Zwischenzeit vollständig geändert. Bei den heutigen großen Anwendungen, in denen verschiedene Chipkarten mit vielen unterschiedlichen Terminals zusammenarbeiten müssen, ist es unabdingbar, dass sich alle eingesetzten Karten elektrisch gleich oder zumindest in klar definierten elektrischen Bereichen einheitlich verhalten.

Die allgemeine internationale Grundlage für die elektrischen Eigenschaften von Chipkarten ist die Norm ISO/IEC 7816-3 und der dazugehörige Anhang ISO/IEC 7816-3 Amd. 1. Der Anhang zur Norm wird bei der nächsten Revision des Hauptteils in diesen integriert und somit als eigenständiges Dokument verschwinden. In dieser Norm sind alle grundlegenden elektrischen Rahmenbedingungen, wie z. B. Spannungsbereiche, maximaler Stromverbrauch und auch Ein- und Ausschaltsequenz von Chipkarten, festgelegt.

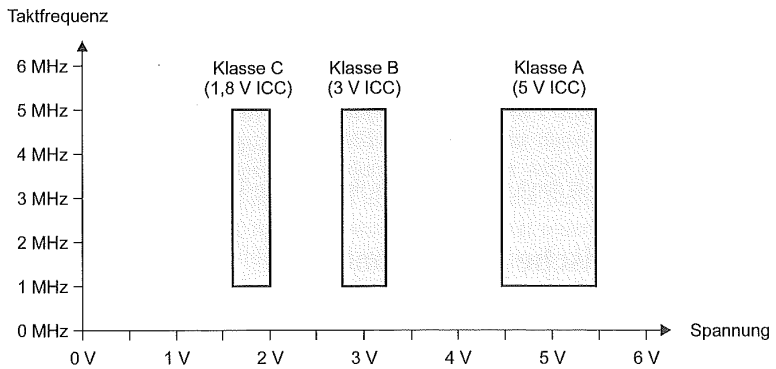


Bild 3.33 Gegenüberstellung der in den drei möglichen Klassen festgelegten Spannungs- und Taktfrequenzbereiche nach ISO/IEC 7816-3 und ISO/IEC 7816-3 Amd. 1.

Wie für eine typische weltweite Norm üblich bietet die ISO/IEC 7816-3 eine Auswahl von Möglichkeiten an, die jedoch in vielen Fällen für die praktischen Anwendungen zu umfangreich sind. Aus diesem Grund konnten sich in Ergänzung zur ISO/IEC-Norm noch im Zahlungsverkehrsbereich der Industriestandard EMV 2000 und im Telekommunikationsbereich die GSM 11.11, GSM 11.12, GSM 11.18 und TS 102.221 etablieren. Beides sind keinesfalls Konkurrenten zur ISO/IEC 7816-3, sondern vervollständigen sie mit sinnvollen Einschränkungen aus der Praxis von Chipkarten-Anwendungen mit Millionen ausgegebener Karten.

Die Unterscheidung zwischen Chipkarten im Zahlungsverkehr und in der Telekommunikation kam deshalb zustande, weil in den beiden Einsatzgebieten einige Anforderungen sich als prinzipiell anders geartet erwiesen. So sind beispielsweise im Bereich des Zahlungsverkehrs die beiden Parameter Stromverbrauch und Spannungsbereich völlig unkritisch, da die verwendeten Terminals fest mit dem Stromnetz verbunden sind. Ganz anders stellt sich die Situation in der Telekommunikation dar. Um eine möglichst lange Einschaltdauer der mit Akkus betriebenen mobilen Endgeräte zu gewährleisten, muss mit jedem Milliwatt gegeizt werden. Deshalb sind gerade hier die Anforderungen an möglichst niedrigen Stromverbrauch und geringe Versorgungsspannung von großer Bedeutung.

Die Tabelle 3.2 zeigt einen Überblick über die wichtigsten elektrischen Parameter der wesentlichen Normen und Industriestandards. Detaillierte Erklärungen zu diesen Parametern finden sich im Anschluss in den folgenden Abschnitten.

Tabelle 3.2 Zusammenfassung der drei elektrischen Parameter Spannung, Strom und Taktfrequenz bei den wichtigsten internationalen Normen für Chipkarten. Die Toleranzangabe für den maximalen Strom sind den jeweiligen Normen zu entnehmen.

Norm und Typ	Spannung	Taktfrequenz	maximaler Strom
ISO/IEC 7816-3 ISO/IEC 7816-3 Amd. 1			
Class A	5 V (± 10 %) ⇒ 4,5 V . . . 5,5 V	1 MHz . . . 5 MHz	60 mA bei 5 MHz
Class B	3 V (± 10 %) ⇒ 2,7 V . . . 3,3 V	1 MHz . . . 5 MHz	50 mA bei 4 MHz
Class C	1,8 V (± 10 %) ⇒ 1,62 V . . . 1,98 V	1 MHz . . . 5 MHz	30 mA bei 4 MHz
Class A, B, C, bei angehaltenem Takt			0,5 mA (*clock stop*)
EMV 2000	5 V (± 10 %) ⇒ 4,5 V . . . 5,5 V	1 MHz . . . 5 MHz	50 mA im gesamten Taktfrequenzbereich
GSM 11.11 5 V SIM	5 V (± 10 %) ⇒ 4,5 V . . . 5,5 V	1 MHz . . . 5 MHz	10 mA im Betrieb (*operating state*) 200 µA bei 1 MHz im Leerlauf (*idle state*) 200 µA bei angehaltenem Takt (*clock stop mode*)
GSM 11.12 3 V SIM	3 V (± 10 %) ⇒ 2,7 V . . . 3,3 V	1 MHz . . . 5 MHz	6 mA bei 3,3 V und 5 MHz im Betrieb (*operating state*) 200 µA bei 1 MHz im Leerlauf (*idle state*) 100 µA bei angehaltenem Takt (*clock stop mode*)
GSM 11.18 1,8 V SIM	1,8 V (± 10 %) ⇒ 1,62 V . . . 1,98 V	1 MHz . . . 5 MHz	4 mA bei 1,8 V und 5 MHz im Betrieb (*operating state*) 200 µA bei 1 MHz im Leerlauf (*idle state*) 100 µA bei angehaltenem Takt (*clock stop mode*)
TS 102.221			
Class A, vom Reset bis zur Auswahl der Anwendung	5 V (± 10 %) ⇒ 4,5 V . . . 5,5 V	1 MHz . . . 5 MHz	10 mA bei 5 MHz im Betrieb (*operating state*) 200 µA bei 1 MHz im Leerlauf (*idle state*)
Class A, während der anwendungs-spezifischen Sitzung			60 mA bei 5 MHz
Class B, vom Reset bis zur Auswahl der Anwendung	3 V (± 10 %) ⇒ 2,7 V . . . 3,3 V	1 MHz . . . 5 MHz	7,5 mA bei 5 MHz im Betrieb (*operating state*) 6 mA bei 4 MHz im Betrieb 200 µA bei 1 MHz im Leerlauf (*idle state*)
Class B, während der anwedungs-spezifischen Sitzung			50 mA bei 5 MHz
Class C, vom Reset bis zur Auswahl der Anwendung	1,8 V (± 10 %) ⇒ 1,62 V . . . 1,98 V	1 MHz . . . 5 MHz	5 mA bei 5 MHz im Betrieb (*operating state*) 4 mA bei 4 MHz im Betrieb 200 µA bei 1 MHz im Leerlauf (*idle state*)
Class C, während der anwendungs-spezifischen Sitzung			30 mA bei 5 MHz

3.3.1 Beschaltung

Chipkarten haben an der Vorderseite entweder acht oder sechs Kontaktflächen. Diese sind die elektrische Schnittstelle zwischen dem Terminal und dem Mikrocontroller der Chipkarten. Alle elektrischen Signale werden über diese Kontakte geführt. Zwei der acht Kontaktflächen (C4 und C8) sind gemäß ISO/IEC 7816-2 für Hilfskontakte AUX1 und AUX2 reserviert und können in Zukunft beispielsweise von der USB-Schnittstelle verwendet werden. Zurzeit haben einige Module für Chipkarten lediglich sechs Kontaktflächen, da dies die Fertigungskosten gegenüber acht Kontaktflächen geringfügig reduziert. Die Funktionalität ist aber identisch mit den Modulen mit acht Kontakten.

Die Kontakte sind analog zu den üblichen Halbleiterbausteinen von links oben nach rechts unten durchnummeriert. Die ISO-Bezeichnung und elektrische Belegung der acht definierten Kontakte ist in Bild 3.34 gezeigt.

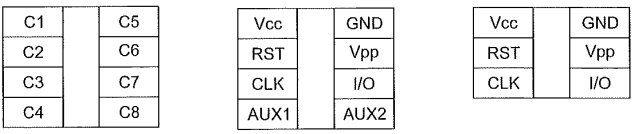


Bild 3.34 Die elektrische Belegung und die Nummerierung der Kontaktflächen einer Chipkarte nach ISO 7816-2.

Bis Ende der Achtzigerjahre war es noch notwendig, die Programmier- und Löschspannung für das EEPROM auf dem Chip extern zuzuführen, da die verwendeten Mikrocontroller über keine Ladungspumpe verfügten. Deshalb wurde für diese Spannung eine Kontaktfläche (C6) reserviert. Da es aber seit Anfang der Neunzigerjahre Stand der Technik ist, diese Spannung mit einer Ladungspumpe direkt auf dem Chip zu erzeugen, wird diese Kontaktfläche nicht mehr verwendet. Allerdings kann sie auch nicht für andere Funktionen benutzt werden, da dies der ISO-Norm widerspräche. So hat jede Chipkarte eine Kontaktfläche, die keine eigentliche Funktion mehr aufweist, aber vorhanden sein muss. Da jedoch der Kontakt für die Programmierspannung zwischen zwei für die Funktion der Chipkarte notwendigen anderen Kontakten liegt, könnte die Kontaktfläche ohnehin nicht eingespart werden, was den Nachteil einer überflüssigen Kontaktfläche etwas mindert.

Tabelle 3.3 Die Bezeichnungen und Funktion der Kontakte nach ISO 7816-2.

Kontakt	Bezeichnung	Funktion
C1	Vcc	Versorgungsspannung (*voltage*)
C2	RST	Eingang Reset (*reset*)
C3	CLK	Eingang für Takt (*clock*)
C4	AUX1	Hilfskontakt 1 (*auxiliary 1*)
C5	GND	Masse (*ground*)
C6	Vpp	Programmierspannung (wird seit langem nicht mehr benutzt)
C7	I/O	Ein-/Ausgang für die serielle Kommunikation (*input/output*)
C8	AUX2	Hilfskontakt 2 (*auxiliary 2*)

3.3.2 Versorgungsspannung

Die Versorgungsspannung einer Chipkarte betrug ursprünglich 5 Volt mit einer maximalen Abweichung von ±10 %. Diese an übliche TTL-Versorgungen angelehnte Spannung war für alle auf dem Markt und in der Anwendung befindlichen Chipkarten der Standardwert.

Analog zu allen anderen Halbleiterbauelementen wurde es mit kleineren Strukturbreiten der Halbleiter und den Anforderungen an reduzierte Stromaufnahme notwendig, den Versorgungsspannungsbereich stark nach unten hin zu erweitern. Verstärkt wurde diese Anforderung vor allem durch den Bereich der Mobiltelefone. Die vom Markt geforderte Gewichtsreduktion der Endgeräte erforderte den Übergang von 6-Volt-Akkus auf 3-Volt-Typen, und da alle Bauteile eines Mobiltelefons in 3-Volt-Technik verfügbar waren, war die Chipkarte zeitweise das einzige Bauteil, das in einem Mobiltelefon noch 5 Volt benötigte. Deshalb benötigten diese zur elektrischen Versorgung der Chipkarte einen aufwendigen und die Kosten unnötig erhöhenden Spannungswandler.

Aus diesen Gründen wurde in der internationalen Normung der Spannungsbereich von Chipkarten in einem ersten Schritt auf 5 Volt bis 3 Volt mit einer Toleranz von ±10 % erweitert. Damit ergibt sich ein effektiver Spannungsbereich von 2,7 Volt bis 5,5 Volt. Jedoch ist bereits absehbar, dass diese Erweiterung unzureichend ist. Deshalb wurde die revidierte ISO/IEC 7816-3 verhältnismäßig zeitnah mit dem Anhang ISO/IEC 7816-3 Amd. 1 um Chipkarten mit 1,8 V Versorgungsspannung und einer Toleranz von ±10 % ergänzt.

Der erweiterte Spannungsbereich stellt weder für einen Mikroprozessor noch für die verschiedenen Speicherarten ein Problem dar, zumal die Kernspannung bei Halbleitern in 0,13 µm-Technologie in der Regel ohnehin nur noch 1,8 V beträgt. Auf Chipkarten-Mikrocontroller ist jedoch ein EEPROM-Speicher integriert. Und genau die für diesen Speichertyp notwendige Ladungspumpe zum elektrischen Löschen von EEPROM-Zellen stellt das größte Hindernis auf dem Weg zu Chipkarten mit niedrigen Versorgungsspannungen dar. Es ist jedoch mit einigem technisch Schaltungsaufwand durchaus möglich, EEPROMs mit dazugehörigen Ladungspumpen auf Mikrocontrollern unterzubringen, die im Bereich zwischen 1,62 Volt und 5,5 Volt funktionsfähig sind.

Die ISO/IEC 7816-3 und der dazugehörige Anhang Amd. 1 sieht zur Kennzeichnung des Spannungsbereichs drei Klassen vor: Class A deckt die Spannung 5 V (±10 %) ab, Class B die Spannung 3 V (±10 %) und Class C die Spannung 1,8 V (±10 %). Alle drei Klassen sind sowohl einzeln als auch in beliebiger Verknüpfung miteinander einsetzbar. Erfüllt eine Chipkarte beispielsweise die Klassen A und B, so kann sie sowohl für 5 Volt als auch für 3 Volt verwendet werden. Dabei ist jedoch zu beachten, dass der Bereich zwischen 4,5 V und 3,3 V außerhalb der Spezifikation liegt und die Chipkarte in diesem Spannungsbereich nicht funktionieren muss. Üblicherweise arbeiten jedoch Chipkarten problemlos zwischen der unteren und oberen Grenze des Spannungsbereichs.

Es gibt noch ein weiteres und ebenfalls sehr wichtiges Kriterium, das die ISO/IEC 7816-3 fordert. Der Mikrocontroller einer Chipkarte darf keinesfalls beschädigt werden, wenn die betreffende Chipkarte mit einer Spannung versorgt wird, die vom Mikrocontroller nicht unterstützt wird. Dieses Kriterium ist gerade zur Sicherstellung der Abwärtskompatibilität neuer Chipkarten zu älteren Terminals unumgänglich. Dies soll den Fall ausschließen, dass beispielsweise die Verwendung einer 3-Volt-Karte in einem 5-Volt-Terminal den Chip auf der Karte zerstört.

Die drei möglichen Spannungsbereiche nach ISO/IEC 7816-3 und ISO/IEC 7816-3 Amd. 1 werden jedoch im Bereich der Zahlungsverkehrs-Chipkarten bisher nicht genutzt. Dort hat sich die ursprüngliche Betriebsspannung der Chipkarten von 5 V ± 10 % bislang behauptet, da stationäre Terminals problemlos und ohne zusätzliche technische Komponenten eine Spannung von 5 V bereitstellen können.

Ganz anders sieht dies bei Chipkarten des Telekommunikationsbereiches aus. Dort sind Chipkarten, die ausschließlich mit 5 V betrieben werden können (*5 V only*) mittlerweile ausgestorben. Als Standardspannung bei GSM hat sich dort seit Ende der Neunzigerjahre 3 V etabliert. Im Bereich der neuen UMTS-Mobilfunknetze zeichnet sich jetzt schon deut-

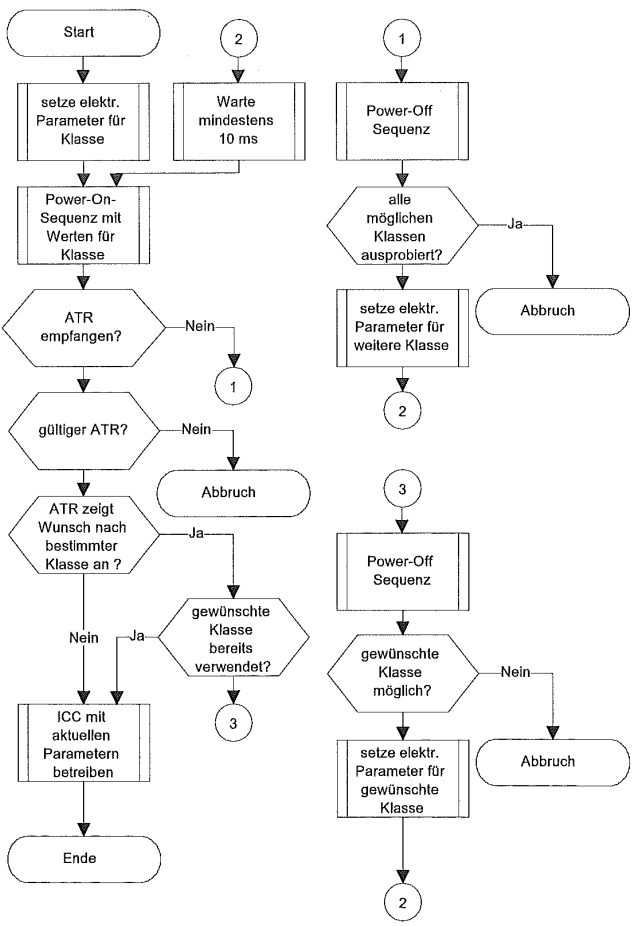


Bild 3.35 Flussdiagramm der Aktionen eines Terminals bei der Auswahl der Versorgungsspannung anhand der in ISO/IEC 7816-3 und ISO/IEC 7816-3 Amd. 1 festgelegten Klassen. Bei mobilen Endgeräten wird üblicherweise der Ablauf immer mit der Klasse der geringsten Versorgungsspannung begonnen.

lich ab, dass es keine Endgeräte mehr gibt, die 5 V unterstützen und lediglich anfangs aus Kompatibilitätsgründen noch eine 3-V-Versorgung vorhanden ist. Mittelfristig wird jedoch 1,8 V die zukünftige Standardspannung sein.

Bei der Auswahl der Versorgungsspannung durch das Terminal legt die ISO/IEC 7816-3 einen bestimmten Ablauf fest. Dieser sieht im Wesentlichen vor, dass die verschiedenen möglichen Spannungen entsprechend den drei Klassen der Reihe nach durchprobiert werden. Sobald das Terminal einen ATR empfangen kann, wird dieser daraufhin analysiert, ob die Chipkarte eine bestimmte Klasse bevorzugt. Ist dies der Fall, initiiert das Terminal eine neue Einschaltsequenz mit der gewünschten Klasse. Enthält der ATR keine Angaben zum Spannungsbereich, so wird die Chipkarte mit demjenigen Spannungswert betrieben, bei dem der erste ATR empfangen werden konnte. Der dazugehörige Ablauf ist in einem Flussdiagramm in Bild 3.35 im Detail beschrieben.
Es ist bereits sicher, dass in Zukunft eine weitere Spannungsklasse eingeführt wird. Diese wird die Betriebsspannung 1,2 V abdecken, welche gerade mit immer kleiner werdenden Strukturbreiten der Halbleiter mittelfristig zu einem typischen Spannungswert für Mikrocontroller avancieren dürfte.

3.3.3 Versorgungsstrom

Der Mikrocontroller der Karte bezieht seine Versorgungsspannung und damit den Versorgungsstrom über den Kontakt C1. Der Strom muss sich dabei unterhalb bestimmter Werte bewegen, damit die entsprechenden Hardwaretreiber seitens der Terminals entsprechend ausgelegt werden können. Die erste Ausgabe der in ISO/IEC 7816-3 im Jahr 1989 legte für den maximalen Stromverbrauch bei 5 V Versorgungsspannung und 5 MHz Taktfrequenz noch einen Wert von 200 mA fest, der jedoch auch damals schon zu hoch gegriffen war. Mittlerweile wurden alle diesbezüglichen Werte erheblich reduziert und auch in Abhängigkeit zu den verschiedenen Klassen für Versorgungsspannungen gesetzt.

Wichtig ist aber auf jeden Fall zu wissen, dass der Stromverbrauch eines Mikrocontrollers direkt proportional sowohl mit der angelegten Versorgungsspannung als auch der angelegten Taktfrequenz ist. Eine weitere, wenn auch nicht so starke Abhängigkeit des Stromverbrauchs besteht noch mit der Temperatur des Mikrocontrollers.

Die aktuelle ISO/IEC 7816-3 sieht für Spannungsklasse A, d. h. für 5 V, einen maximalen Vorsorgungsstrom von 60 mA bei der maximalen Taktfrequenz von 5 MHz und höchstens 50 Grad Celsius Umgebungstemperatur vor.

Im Bereich der Zahlungsverkehrs-Chipkarten wurden in der EMV 2000-Spezifikation der maximale Strom der ISO/IEC 7816-3 Norm von 60 mA auf 50 mA reduziert, jedoch ansonsten keine signifikanten Ergänzungen vorgenommen. Im Telekommunikationsbereich ist der Stromverbrauch seit den Ursprüngen eine kritische Größe. Deshalb existiert dort mittlerweile ein kompliziertes Regelwerk, in dem abhängig von Taktfrequenz und Zustand der Chipkarte der maximale Strom angegeben ist. Eine detaillierte Aufstellung der maximalen Ströme in Abhängigkeit der Spannungsklassen zeigt Tabelle 3.2.

Bei Chipkarten entsprechend der USIM-Spezifikation wurde eine technisch interessante Neuerung eingeführt. Dort wurden für den Stromverbrauch zwei unterschiedliche Betriebszustände der Chipkarte festgelegt. Der erste Zustand umfasst die Zeitspanne vom Reset bis zur Auswahl der Anwendung und der zweite Zustand schließt die darauf folgende

anwendungsspezifische Sitzung ein. Der maximal erlaubte Stromverbrauch ist im ersten Zustand deutlich niedriger bemessen als im zweiten Zustand. Zudem hat das Endgerät die Möglichkeit, über ein anwendungsspezifisches Datenobjekt herauszufinden, welchen Strombedarf die jeweils selektierte Anwendung hat. Der Hintergrund dabei ist, dass Chipkarten bei der Aktivierung des numerischen Koprozessors oder eines internen Frequenzvervielfachers zur Leistungssteigerung des Prozessors einen erheblich höheren Stromverbrauch haben. Mit diesem Mechanismus wäre es zumindest prinzipiell möglich, dass ein Mobiltelefon nur die Chipkarten-Anwendungen verwendet, die es auch hinsichtlich des Strombedarfs ausreichend unterstützen kann. Leider enthält die entsprechende USIM-Spezifikation kein Verfahren, um den maximal zu liefernden Strom zwischen Mobiltelefon und Chipkarte ähnlich einem PPS auszuhandeln. Beim jetzigen Stand der Spezifikation bleibt deshalb dem Mobiltelefon bei der Anforderung eines zu großen Strombedarfs seitens der Chipkarte keine andere Möglichkeit, als diese abzuschalten.

Moderne Mikrocontroller für Chipkarten haben einen Stromverbrauch in der Größenordnung von etwa 350 µA pro MHz Takt. Mit diesen Werten kann man die folgende Formel für den Stromverbrauch eines Mikrocontrollers in Abhängigkeit vom angelegten oder im Chip erzeugten Takt aufstellen. Die erhaltenen Werte lassen sich für erste Abschätzungen gut verwenden, es ist jedoch zu beachten, dass der Stromverbrauch nicht nur von der Taktfrequenz, sondern selbstverständlich auch von der angelegten Versorgungsspannung, der Temperatur und natürlich vom Chiptyp abhängig ist.

$$I = \frac{f}{2{,}857} \cdot \frac{\text{mA}}{\text{MHz}}$$

Bei einer Versorgungsspannung von 5 Volt und einem angenommenen Stromverbrauch von 60 mA hat eine Chipkarte eine Leistungsaufnahme von 300 mW. Dieser Wert ist noch so niedrig, dass man sich über eine gefahrbergende Eigenerwärmung des Chips während des Betriebes keine Sorgen zu machen braucht, auch wenn diese Energiemenge auf einer Fläche von etwa 20 mm^2 umgesetzt wird.

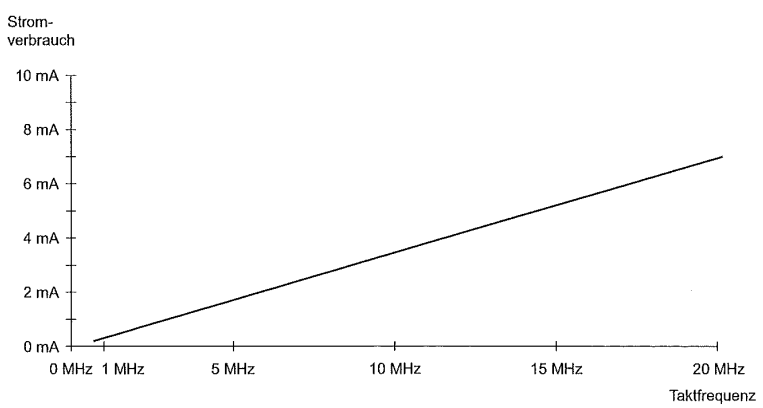


Bild 3.36 Abhängigkeit des Stromverbrauchs eines Chipkarten-Mikrocontrollers vom angelegten Takt. Der Chip befindet sich im normalen Betrieb und nicht im stromsparenden Sleep-Modus. Der Stromverbrauch im Sleep-Modus ist bei angelegtem Takt ebenfalls linear von der angelegten Taktfrequenz abhängig und beträgt bei 5 MHz je nach Mikrocontroller ca. 50 µA.

Alle Mikrocontroller für Chipkarten besitzen einen oder mehrere spezielle Strom-Spar-Modi. Das Funktionsprinzip beruht darauf, alle nicht benutzten Baugruppen auf dem Chip zu deaktivieren. Prinzipiell muss nur noch die Interruptlogik der I/O-Schnittstelle, die Prozessorregister sowie das RAM bestromt werden, der Rest des Mikrocontrollers kann deaktiviert sein. Das RAM muss deshalb seinen Inhalt während der Sitzung erhalten, um die aktuellen Zustände zu speichern. In der Praxis wird oft auch noch der Prozessor versorgt, aber ROM und EEPROM sind ausgeschaltet. Befindet sich der Mikrocontroller in diesem auch Sleep-Modus oder *idle state* genannten Zustand, so sinkt der Stromverbrauch erheblich, da die meisten Teile des Chips dann von der Versorgungsspannung abgetrennt sind. Viele Chipkarten-Mikrocontroller unterstützen in Erweiterung des Sleep-Modus noch einen Modus, bei dem auch der angelegte Takt abgeschaltet werden kann. Dies wird *clock stop mode* genannt. Damit können dann vor allem im Terminal die Hardware-Komponenten für die Takterzeugung deaktiviert werden, weshalb dieser Modus besonders für batteriebetriebene Endgeräte im Mobilfunkbereich von Interesse ist.

Nach ISO/IEC 7816-3 ist der maximale Strom, der im Sleep-Modus mit angehaltenem Takt verbraucht werden darf, mit 500 µA für alle 3 Klassen bei angehaltenem Takt angegeben. Für den Mobilfunkbereich ist dieser Wert bereits zu hoch. So legt beispielsweise die GSM 11.11 für 5-V-Chipkarten eine Obergrenze von 200 µA bei 1 MHz Takt fest.

Im Zusammenhang mit dem Versorgungsstrom existiert noch ein wichtiges Detail, dessen Missachtung Terminalherstellern immer wieder arges Kopfzerbrechen bereitet. Alle verwendeten Mikrocontroller basieren auf der CMOS-Technologie. Daher können bei Schaltvorgängen von Transistoren unter Umständen hohe Querströme auftreten. Diese Querströme verursachen Spikes mit einem Vielfachen des Nennstromes im Nanosekundenbereich. Auch durch das Anschalten der Ladungspumpe für das EEPROM können diese Spikes entstehen. Kann dieser hohe Strom vom Terminal in der kurzen Zeit nicht aufgebracht werden, so fällt die Versorgungsspannung unzulässig weit ab. Dies kann zu einem Schreibfehler im EEPROM führen oder zum Auslösen der Unterspannungsdetektion im Chip.

Das Vorhergehende ist der Grund dafür, warum mittlerweile in praktisch jeder Norm und Spezifikation ein Hinweis auf diese möglichen Spikes zu finden ist. So fordert beispielsweise die ISO/IEC 7816-3 für die Spannungsklasse A (d. h. 5 V), dass die Stromversorgung fähig sein muss, Spikes mit einer maximalen Dauer von 400 ns und einer maximalen Amplitude von 100 mA abzufangen. Das entspricht bei einem angenommenen dreieckförmigen Spike einer bereitzustellenden Ladung von 20 nAs. Durch Anbringung eines 100 nF Keramikkondensators zwischen Masse und Versorgungsspannung sehr nahe an den Kontakten zur Chipkarte kann das Problem auf einfache Weise gelöst werden.

3.3.4 Taktversorgung

Chipkartenprozessoren verwenden in der Regel keine interne Takterzeugung. Deshalb muss von außen ein Takt angelegt werden, der gleichzeitig auch die Referenz für die Geschwindigkeit der Datenübertragung ist. Das Tastverhältnis des Takts sollte dabei nach ISO/IEC 7816-3 und den meisten anderen Normen und Spezifikationen 50 % betragen.

Der übliche Toleranzbereich für die maximale Abweichung ist ein Tastverhältnis zwischen 40 % und 60 %.

Der an die Kontaktfläche angelegte Takt ist nicht unbedingt identisch mit dem intern dem Prozessor zur Verfügung gestellten. Bei einigen Mikrocontrollern kann optional ein Taktvervielfacher oder ein Teiler zwischen externem und internem Takt eingefügt werden. Oft hat der Teiler den Wert zwei, sodass der externe Takt doppelt so hoch ist wie der interne. Die Gründe dafür liegen einerseits bei der Chiphardware und andererseits in der somit geschaffenen Möglichkeit, im Terminal schon vorhandene Oszillatoren für die Takterzeugung der Chipkarte zu nutzen.

Die meisten Chipkarten-Mikrocontroller erlauben ein Abschalten der Taktversorgung, falls sich die CPU im Sleep-Modus befindet. Abschalten bedeutet dabei ein Einfrieren der Taktversorgung auf einen bestimmten Pegel. Je nach Halbleiterhersteller kann der bevorzugte Abschaltpegel high oder low sein. Die Stromaufnahme der Chipkarte an der Taktversorgung beträgt nur wenige Mikroampere, deshalb mag das Abschalten des Taktes auf den ersten Blick verwunderlich erscheinen. Allerdings existiert ein Stromspareffekt bei der Takterzeugung im Terminal, sodass es sich je nach Anwendung durchaus lohnen kann.

3.3.5 Datenübertragung

Bei Fehlern während der Datenübertragung kann es unter Umständen vorkommen, dass sowohl Terminal als auch Chipkarte gleichzeitig senden. Dies verursacht auf der verbindenden I/O-Leitung eine Datenkollision. Unabhängig vom Problem auf der Anwendungsebene kann dies auf der physikalischen Schicht zu mehr oder minder großen Strömen auf der I/O-Leitung führen, die die Schnittstellenbausteine zerstören können.

Um für diesen Fall Beschädigungen an den Halbleitern zu vermeiden, wird die I/O-Leitung im Terminal mit einem 20 kΩ Pull-Up-Widerstand auf +5 Volt Pegel gelegt. Mit der zusätzlichen Vereinbarung, nie einen aktiven 5 Volt Pegel zu senden, wird vermieden, dass die Kommunikationspartner im Fehlerfall mit unterschiedlichen Pegeln gegeneinander treiben. Muss also im Laufe der Kommunikation die I/O-Leitung auf +5 Volt Pegel gebracht werden, so schaltet der jeweilige Partner seinen Ausgang in den hochohmigen Zustand (Tri-State), und durch den Pull-Up-Widerstand wird die Leitung von selbst auf den +5 Volt Pegel hochgezogen.

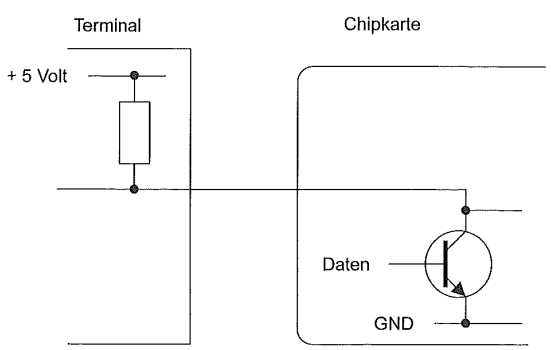


Bild 3.37 Die Beschaltung der I/O-Leitung zwischen Terminal und Chipkarte.

3.3.6 An-/Abschaltsequenz

Gegenüber Ladungen und Spannungen an den Kontakten sind alle Chipkarten-Mikrocontroller abgesichert. Um undefinierte Zustände zu vermeiden, ist eine genau festgelegte An- und Abschaltsequenz vorgeschrieben und muss auch strikt eingehalten werden. Dies spiegelt sich auch im entsprechenden Teil der ISO/IEC 7816-3. Die Sequenzen definieren den elektrischen Teil des An- und Abschaltens, sie haben nichts mit der Reihenfolge der mechanischen Kontaktierung zu tun, die im Übrigen nicht festgelegt ist. Klugerweise verwendet man aber trotzdem bei der Kontaktierung einen vorauseilenden Kontakt für die Masse, sodass auch hier elektrisch definiert kontaktiert und später dekontaktiert wird.

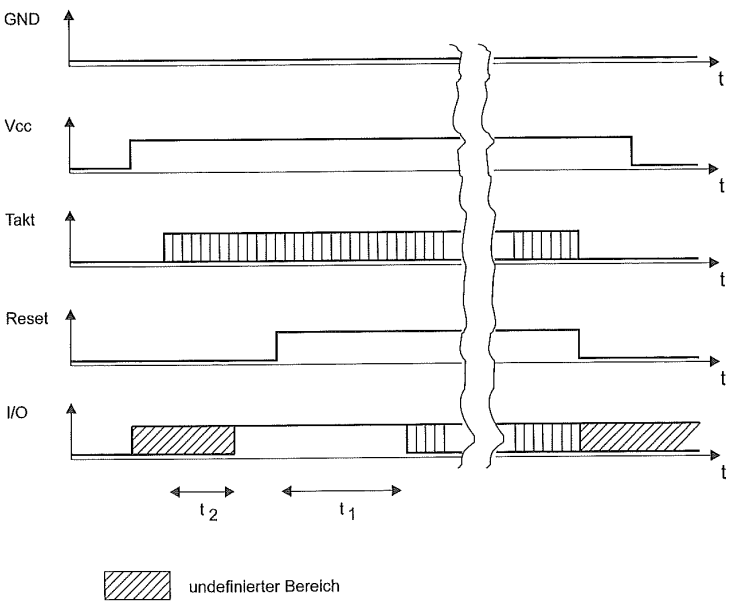


Bild 3.38 An- und Abschaltsequenz einer Chipkarte nach ISO/IEC 7816-3. Die Zeiten t_1 und t_2 sind in den Bereichen: $(400/f) \leq t_1 \leq (40\,000/f)$ und $t_2 \leq (200/f)$.

Das Bild 3.38 zeigt, dass zuerst Masse angelegt werden muss und erst danach die Versorgungsspannung. Anschließend folgt der Takt. Würde man beispielsweise vor der Versorgungsspannung den Takt anlegen, so versucht der Mikrocontroller, seinen gesamten Versorgungsstrom über die Taktversorgung zu ziehen. Dabei kann der Chip irreversibel beschädigt werden und ist dann nicht mehr funktionsfähig. Die gleichen Auswirkungen auf den Mikrocontroller wären auch bei einer nicht eingehaltenen Abschaltsequenz zu befürchten.

Ist der Mikrocontroller in Betrieb, kann er über die Reset-Leitung zurückgesetzt werden. Dazu ist zuerst ein Low-Pegel auf dieser Leitung notwendig; mit der steigenden Flanke wird dann der Reset eingeleitet. Dieser Reset während des Betriebs wird analog zu anderen Computersystemen als Warmreset bezeichnet. Ein Kaltreset ist demgegenüber ein Reset, der über das ISO-gerechte Anschalten aller Versorgungsleitungen zustande kommt.

3.4 Mikrocontroller für Chipkarten

Aus informationstechnischer Sicht ist der zentrale Bestandteil einer Chipkarte der unter den Kontaktflächen eingebettete Mikrocontroller. Er steuert, initiiert und überwacht alle Aktivitäten. Die speziell für diese Belange abgestimmten und entwickelten Mikrocontroller sind vollständige Computer. Das heißt, sie sind mit einem Prozessor, Speicher und einer Schnittstelle zur äußeren Welt ausgestattet.

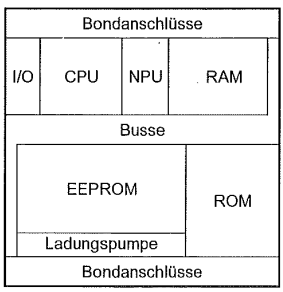


Bild 3.39 Mögliche Anordnung der wesentlichen Funktionseinheiten auf einem Die bei einem einfachen Chipkarten-Mikrocontroller.

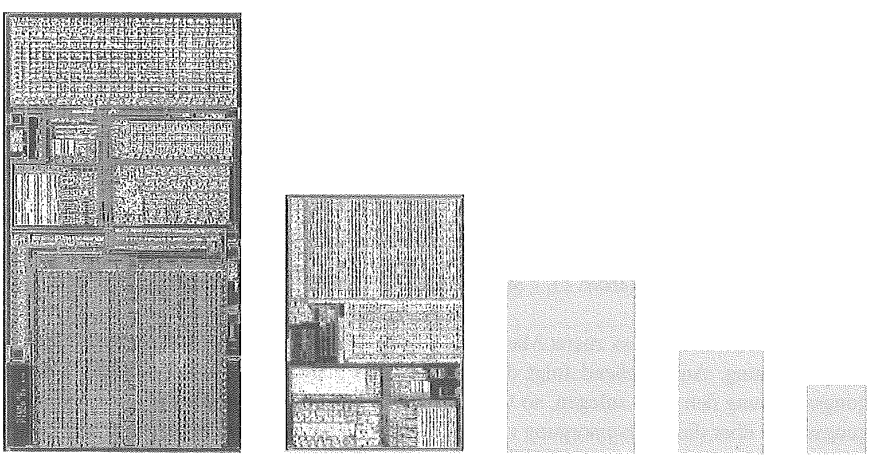


Bild 3.40 Größenvergleich von zwei funktional vollständig identischen Chipkarten-Mikrocontrollern vor und nach einer Chipflächenreduzierung („Shrink-Prozess"). Ganz links ist ein SLE 44C80 in 1 µm-Technologie mit 21,7 mm² Fläche abgebildet und rechts daneben ein SLE 44C80S in 0,8 µm-Technologie mit 10 mm² Fläche. Die drei grauen Rechtecke im Anschluss daran zeigen die jeweiligen Größenverhältnisse an, wenn man diesen Chipkarten-Mikrocontroller bei entsprechender Größenreduzierung in 0,5 µm, 0,35 µm bzw. 0,13 µm-Technologie fertigen würde. (Fotos: Infineon)

Die wichtigsten funktionellen Bausteine eines typischen Chipkarten-Mikrocontrollers sind: der Prozessor, der Adress- und Datenbus und die drei verschiedenen Speicherarten (RAM, ROM, EEPROM). Zusätzlich befindet sich noch eine Schnittstellenbaugruppe auf dem Chip, die die serielle Kommunikation mit der äußeren Welt übernimmt. Diese Schnittstelle darf man sich allerdings nicht als komplexe Funktionseinheit auf dem Chip vorstellen, die autark senden und empfangen kann. Serielle Schnittstelle bedeutet in diesem Bereich im einfachsten Fall nur eine von der CPU ansprechbare Adresse, die als I/O-Leitung auf die Kontaktfläche herausführt.

Zusätzlich bieten manche Hersteller noch spezielle Rechenwerke auf dem Chip an, die als eine Art mathematischer Koprozessor fungieren. Die Funktionalität dieser Recheneinheiten umfasst allerdings nur Exponential- und Modulo-Operationen mit Ganzzahlen. Beide Operationen sind elementare Bestandteile und Notwendigkeit von Public-Key-Verschlüsselungsverfahren, wie z. B. dem RSA-Algorithmus.

Die zurzeit üblicherweise eingesetzte Halbleitertechnologie bei Chipkarten-Mikrocontrollern bewegt sich in der Größenordnung 0,25 µm, 0,18 µm und 0,13 µm und befindet sich damit durchaus im Bereich der kleinsten derzeit technisch machbaren Strukturbreiten.

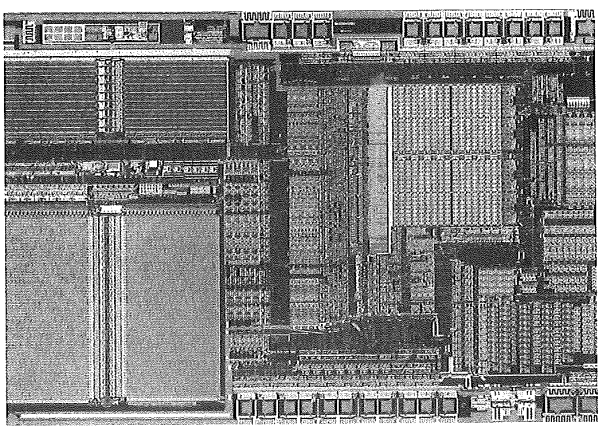


Bild 3.41 Foto des Chipkarten-Mikrocontrollers P 83 C 852 mit den Funktionseinheiten ROM, EEPROM, Prozessor mit Koprozessor und RAM (von links oben nach rechts unten). Der Chip hat eine Fläche von 22,3 mm^2 und besitzt 183 000 Transistoren. Dieser Chip wird mittlerweile nicht mehr produziert, zeigt jedoch sehr gut die Anordnung der einzelnen Funktionseinheiten auf dem Die. (Philips)

Die bei Chipkarten verwendeten Mikrocontroller sind keine Standardbauelemente, die man überall kaufen kann. Sie sind speziell für diesen Einsatzzweck entwickelte Bausteine und werden auch nicht für andere Anwendungszwecke verwendet. Dafür gibt es einige gewichtige Gründe:

Herstellungskosten

Die benötigte Fläche für den Mikrocontroller auf dem Silizium-Wafer ist eine der entscheidenden Einflussgrößen bei den Herstellungskosten. Auch verursachen großflächige

Chips eine aufwendigere und somit teure Verpackung in Module. Deshalb versucht man die Chipfläche so weit wie möglich zu reduzieren.

Weiterhin sind viele auf dem Markt erhältliche Standardbauelemente mit technischen Funktionen ausgestattet, die für Chipkarten nicht benötigt werden. Da diese Funktionen zusätzliche Flächen auf dem Silizium belegen, können sie bei Chips, die speziell für Chipkarten gestaltet sind, gestrichen werden. Durch diese Flächenoptimierung lassen sich die Herstellungskosten für einen einzelnen Chip zwar nur minimal senken, doch durch die großen Stückzahlen summieren sich die kleinen Einsparungen zu signifikanten Beträgen, die eine Anpassung des Chip-Designs rechtfertigen.

Funktionalität

Aufgrund der Bedingung, alle funktionellen Komponenten eines Computers auf einem einzelnen Silizium-Chip zu integrieren, ist das Angebot der in Frage kommenden Halbleiterbauelemente äußerst eingeschränkt. Unter der Vorgabe einer minimalen Chipfläche, 5 Volt oder 3 Volt Spannungsversorgung und einer seriellen Schnittstelle auf dem Chip scheiden dann im Wesentlichen alle Standardbauelemente aus. Zusätzlich muss sich auf dem Chip auch noch ein Speicher (EEPROM oder Flash-EEPROM) befinden, der gelöscht und beschrieben werden kann, aber zur Speicherung von Daten keine permanente Spannungsversorgung benötigt.

Sicherheit

Da Chipkarten meist in sicherheitsrelevanten Bereichen eingesetzt werden und damit auf dem Chip sowohl passive als auch aktive Schutzmaßnahmen notwendig sind, ergibt sich zwangsläufig die Forderung nach einem speziell für diesen Einsatzzweck entwickelten Chip.

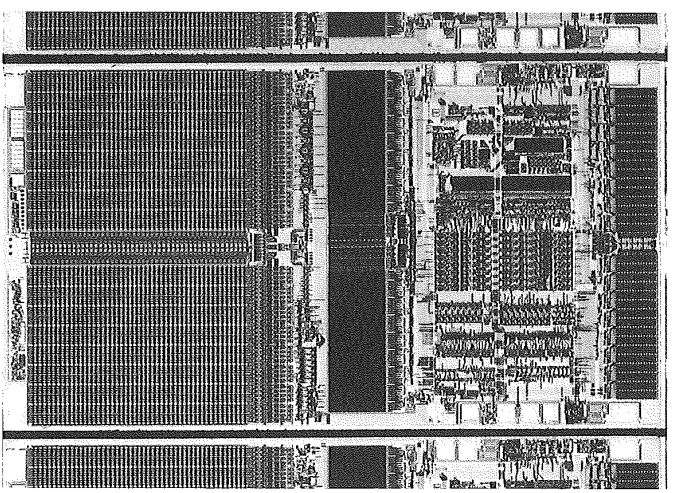


Bild 3.42 Chipkarten-Mikrocontroller ST 16 623 mit den Funktionseinheiten (von links nach rechts) EEPROM, ROM, Prozessor und RAM. Dies Chip wird mittlerweile nicht mehr produziert, zeigt jedoch sehr gut die Anordnung der einzelnen Funktionseinheiten auf dem Die. (ST Microelectronics)

Chipfläche
Die Fläche, die der Mikrocontroller auf dem Silizium einnimmt, beeinflusst stark die Bruchempfindlichkeit des Chips. Je größer diese Fläche, desto leichter kann bei einer Biege- oder Torsionsbeanspruchung der Karte ein Chipbruch eintreten. Man denke nur an die in einer Geldbörse mitgeführte Telefonkarte. Die Beanspruchung der Karte und des in ihr eingebetteten Chips durch Biegung ist dabei enorm. Selbst feinste Haarrisse im Chip reichen aus, die Chipkarte funktionsunfähig zu machen. So ist von den meisten Kartenherstellern eine Obergrenze der Chipfläche von ca. 25 mm^2 in einem möglichst quadratischen Chiplayout festgelegt, um die Bruchgefahr so weit wie möglich zu minimieren.

Verfügbarkeit
Manche Kartenhersteller vertreten die Sicherheitsphilosophie, dass die eingesetzten Chipkarten-Mikrocontroller nicht auf dem freien Markt verfügbar sind. Dadurch erschwert man Analysen der Chiphardware beträchtlich, da ein Angreifer normalerweise keinen Zugang hat. Dieses Argument ist jedoch mit der allgemeinen Verfügbarkeit von frei programmierbaren Chipkarten wie beispielsweise Java Card stark in den Hintergrund getreten und im Allgemeinen für Standardanwendungen heute hinfällig.

Die Spezialisierung auf einige wenige Mikrocontroller und somit Hersteller hat aber den Nachteil, dass die Abhängigkeit der Kartenhersteller sehr groß ist. Im Falle von Produktionsengpässen bei einem Halbleiterhersteller ist es dann nicht möglich, schnell auf andere Bausteine zu wechseln.

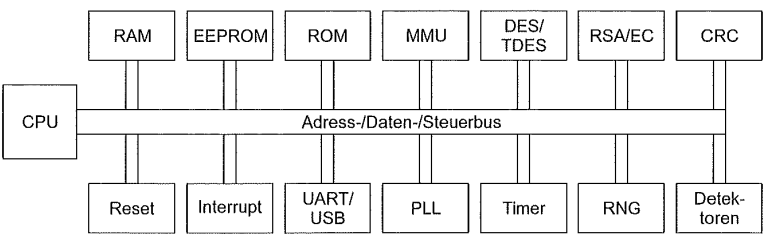


Bild 3.43 Die wesentlichen logischen Funktionseinheiten eines leistungsfähigen Chipkarten-Mikrocontrollers.

3.4.1 Prozessortypen

Die in Chipkarten eingesetzten Prozessoren sind keine Spezialentwicklungen, sondern in anderen Bereichen bewährte und seit langem eingesetzte Bauelemente. Es ist nämlich in der Branche nicht üblich, Prozessoren für spezielle Einsatzgebiete neu zu entwickeln, da dies im Allgemeinen zu teuer ist. Auch hätte man dann einen völlig unbekannten Prozessor, für den es bei den Betriebssystemherstellern keine entsprechenden Funktionsbibliotheken und Entwicklungswerkzeuge gibt.

Zudem müssen Prozessoren für Chipkarten extrem zuverlässig sein, weswegen man älteren und damit in der Praxis bewährten Prozessoren wesentlich mehr vertraut als den jeweils neuesten Entwicklungen der Halbleiterhersteller. In den sehr auf Funktionssicherheit bedachten Bereichen der Luftfahrt- und Weltraumtechnik setzt man aus den gleichen

Gründen auch nur Bausteine ein, die ein oder zwei Generationen hinter dem aktuellen Stand der Technik sind.

Im Jahr 1947 wurde der Transistor bei Bell erfunden, und im Jahr 1971 brachte Intel den ersten Mikroprozessor auf den Markt. Er hatte die Bezeichnung 4004, 2 300 Transistoren, 108 kHz Taktfrequenz, 45 Maschinenbefehle, 640 Byte Adressraum, 4 bit Datenbusbreite und daraus resultierenden 0,06 MIPS Rechenleistung. Seitdem hat es bei diesen Bausteinen eine immense Weiterentwicklung gegeben. Neuere Produkte wie der 32 Bit Pentium IV Prozessor mit 42 Millionen Transistoren auf 217 mm^2 in 0,13 µm-Technologie und 2 GHz Taktfrequenz zeigen dies deutlich. Im Bereich der Chipkarten werden aber aus obigen Gründen nicht die technologisch fortschrittlichsten Prozessorbausteine eingesetzt. Eine Transistorzahl von 200 000 repräsentiert in etwa einen Chip in der mittleren Leistungsklasse.

Chipkarten-Mikrocontroller am unteren Ende der Leistungsskala haben einen adressierbaren Speicher, der sich zwischen 6 kByte und 30 kByte bewegt. Unter diesen Rahmenbedingungen ruft es keine Einschränkungen hervor, wenn Prozessoren mit einer Speicherbreite von 8 Bit verwendet werden. Die Prozessoren selber basieren in der Regel auf einer CISC (*complex instruction set computer*) -Architektur. Sie benötigen also für jeden Maschinenbefehl mehrere Takte und haben meist einen sehr umfangreichen Befehlssatz. Der Adressbereich der 8-Bit-Prozessoren ist durchweg 16 Bit, mit denen sich maximal 65 536 Byte adressieren lassen. Die Befehlssätze der Prozessoren orientieren sich entweder an der Motorola 6805 oder an der Intel 8051-Architektur. Zum Teil sind die vorhandenen Befehle je nach Halbleiterhersteller durch Erweiterungen ergänzt. Diese betreffen meistens zusätzliche Möglichkeiten der 16-Bit-Adressierung der Speicher, die bei den beiden als Grundlage dienenden Befehlssätzen nur in sehr rudimentärer Form vorhanden sind.

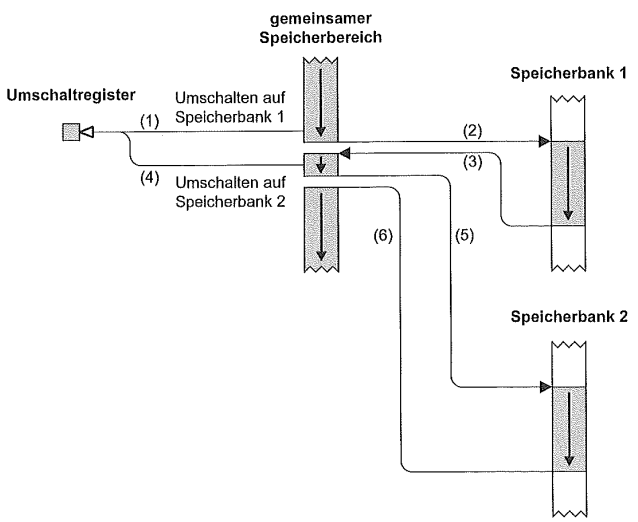


Bild 3.44 Prinzipieller Aufbau und Ablauf im Programmfluss bei der Benutzung eines in mehrere Speicherbänke aufgeteilten Speichers. Das Beispiel zeigt den Aufruf von zwei Unterroutinen, die sich in zwei verschiedenen Speicherbänken befinden, von einem gemeinsamen Speicherbereich aus. Die geklammerten Zahlen kennzeichnen die Reihenfolge des Ablaufs.

Ebenfalls verfügen die 8-Bit-Prozessoren bei Bedarf über Erweiterungen hinsichtlich der Adressierung zusätzlicher Speicherbänke zur Überwindung der 64-kByte-Grenze. Der Zugriff auf diese Speicherbänke wird über spezielle Register gesteuert, um sie so für den Zugriff durch den Prozessor in einen bestimmten Adressbereich einzublenden. Diese nichtlineare Adressierung des Speichers hat jedoch erhebliche Nachteile. So erhöht sie durch die verhältnismäßig komplizierte Aufteilung des Programmcodes auf die verschiedenen Speicherbänke in erheblichem Maße die Komplexität der Software und damit die Fehlerwahrscheinlichkeit. Zusätzlich ist auch Speicher für die Umschaltfunktionalität benötigt, weshalb diese Adressraumerweiterungen über Speicherbänke vor allem ein Notbehelf in Richtung Übergang auf Prozessoren mit größerer Bitbreite ist.

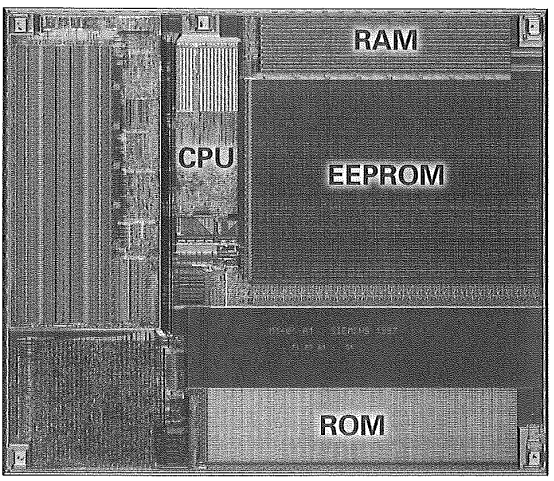


Bild 3.45 21 mm² großer Chipkarten-Mikrocontroller SLE 66CX160S in 0,6 µm-Technologie mit 32 kByte ROM, 16 kByte EEPROM und 1280 Byte RAM. Die beiden nicht beschrifteten Bereiche auf der linken Seite des Chips sind der numerische Koprozessor sowie Peripheriebaugruppen (Timer, Zufallszahlengenerator, CRC-Rechenwerk). Auf dem Foto sind am Chiprand sehr gut die fünf Bond-Pads für die elektrische Verbindung mit den Modulkontakten zu erkennen. (Infineon)

Bei den 16-Bit-Prozessoren existieren zum Teil Weiterentwicklungen bestehender 8051-Architekturen als auch firmenspezifische Konzepte wie Hitachi H8, Philips XA oder Samsung CALM. Der H8 war im Übrigen lange Zeit der einzige 16-Bit-Prozessor für Chipkarten-Mikrocontroller auf der Grundlage einer RISC (*reduced instruction set computer*) -ähnlichen Architektur und eines entsprechenden Befehlssatzes.

Am oberen Ende der Leistungsskala der Chipkarten-Mikrocontroller sind mittlerweile sowohl 16-Bit-Typen als auch 32-Bit-Typen verfügbar. Die Entwicklungsrichtung weist ziemlich deutlich in Richtung 32-Bit-Prozessoren, was in dieser Leistungsklasse auch dringend notwendig ist, um die großen Speicher jenseits der 64-kByte-Grenze zu verwalten und vor allem den enormen Leistungshunger der modernen Interpreter gestützten Chipkarten-Betriebssystemen wie Java Card zu befriedigen. Wichtige Kriterien für die Auswahl der

Prozessoren bilden Aspekte wie Code-Dichte, Leistungsaufnahme und Resistenz gegenüber Angriffen.

Bei den 32-Bit-Prozessoren etablieren sich zurzeit ebenfalls firmenspezifische Typen, doch haben auch zwei Prozessorkerne aus typischen Controllermärkten Einzug in die Chipkartenwelt gehalten. Es sind dies MIPS- [MIPS] und ARM-Prozessoren [ARM].

Der erste Schritt in die 32-Bit-Gefilde wurde ab 1993 im europäischen CASCADE-Projekt (*Chip Architecture for Smart Card and portable intelligent Devices*) getan. Das Ziel war unter anderem, einen leistungsfähigen Prozessor für Chipkarten bereitzustellen. Die Wahl fiel auf den RISC-Prozessor ARM 7M, der häufig in tragbaren Geräten (z. B.: Videokameras, PDAs u. Ä.) eingesetzt wird.[3] Er verfügt über eine 32-Bit-Architektur, kann bis 20 MHz bei 3 Volt Versorgungsspannung betrieben werden und benötigt dann maximal 40 mA. In 0,8 µm-Technologie nimmt der ARM 7 Kern eine Fläche von 5,9 mm^2 ein (3,12 mm × 1,90 mm), und die dazugehörige Recheneinheit für die üblichen Public-Key-Algorithmen (RSA, DSA, EC) benötigt noch zusätzliche 2 mm^2. Der Prozessor unterstützt durch seinen Supervisor- und User-Mode auch eine Separierung des Programmcodes von Chipkarten-Betriebssystem und Anwendung. Der ARM 7 wurde mittlerweile für Chipkarten optimiert und wird als SC 100 (*secure core*) von verschiedenen Halbleiterfirmen in ihre Chipkarten-Mikrocontroller eingebaut. Der nächste Evolutionsschritt ist eine dedizierte Chipkartenvariante des ARM 9 mit der Bezeichnung SC 200. Ähnlich verhält es sich mit den MIPS-Prozessoren, die ursprünglich ebenfalls für andere Einsatzgebiete entwickelt wurden.

Obwohl die 32-Bit-Prozessoren durch ihre breiteren Busstrukturen und aufwendigeren internen Komponenten deutlich mehr Fläche in gleicher Halbleitertechnologie als 8-Bit-CPUs auf dem Die benötigen, werden sie in Zukunft immer mehr Einzug in Chipkarten halten. Die von ihnen zur Verfügung gestellte Rechenleistung ist für zukünftige Chipkarten-Anwendungen zwingend erforderlich, sodass die Nachteile der größeren Leistungsauf-

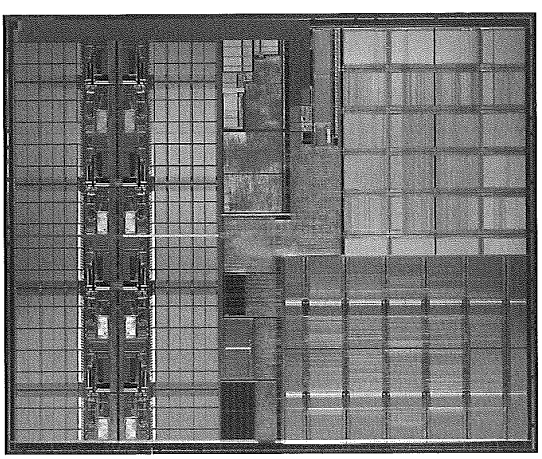


Bild 3.46 Chipkarten-Mikrocontroller SLE 88CX720P in 0,22 µm-Technologie mit 240 kByte ROM, 80 kByte EEPROM und 8 kByte RAM. (Infineon)

[3] nach [Peyret 97]

nahme und des erhöhten Flächenbedarfs auf dem Chip dafür in Kauf genommen werden können. Allerdings wird es wohl auf absehbare Zeit nicht dazu kommen, dass die 8-Bit-Prozessoren aussterben, da diese eine solide Grundlage für kostengünstige Chips der unteren Leistungsklassen darstellen.

3.4.2 Speicherarten

Neben dem Prozessor sind die verschiedenen Speicher die wichtigsten Bestandteile eines Mikrocontrollers. Sie dienen zur Ablage von Programmcode und Daten. Weil Chipkarten-Mikrocontroller vollständige Computer sein müssen, ergibt sich die charakteristische Aufteilung des Speichers in RAM, ROM und EEPROM. Die Aufteilung hängt sehr stark vom späteren Anwendungsgebiet des Chips ab. Allerdings wird immer versucht, RAM- und EEPROM-Speicher so klein wie möglich zu halten, da diese am meisten Platz pro Bit benötigen.

Für Chipkarten, die mehrere Anwendungen gleichzeitig verwalten können, d. h. Multiapplication-Chipkarten, benutzt man meist Chips, bei denen das ROM etwa doppelt so groß wie das EEPROM ist, um dort das aufwendige Betriebssystem unterzubringen. Für Chipkarten, die nur für eine einzige Anwendung verwendet werden, wählt man Mikrocontroller aus, dessen EEPROM-Speicher nur etwas größer ist als die Daten der Anwendung. Auf diese Weise können alle variablen Anwendungsdaten einschließlich einiger Teile des Betriebssystems im EEPROM gespeichert werden, wobei das auf dem Silizium sehr flächenintensive und darum teure EEPROM bestmöglich ausgenutzt ist.

Die Integration von drei verschiedenen Speicherarten auf einem einzigen Stück Silizium ist halbleitertechnisch aufwendig. Es erfordert eine erhebliche Anzahl von Prozessschritten und Belichtungsmasken. Der Platzbedarf der einzelnen Speichertypen ist durch den unterschiedlichen Aufbau und die Funktionsweise auch sehr unterschiedlich. So benötigt eine RAM-Zelle etwa 4-mal mehr Platz als eine EEPROM-Zelle und diese wiederum 4-mal so viel wie eine ROM-Zelle. Dies ist auch der Grund, warum Chipkarten-Mikrocontroller so sparsam mit RAM ausgestattet sind. RAM-Speicher mit 4 kByte Größe gelten schon als groß. Bedenkt man, dass auf der gleichen Fläche 16 kByte EEPROM oder 64 kByte ROM untergebracht werden könnten, so ist dies verständlich.

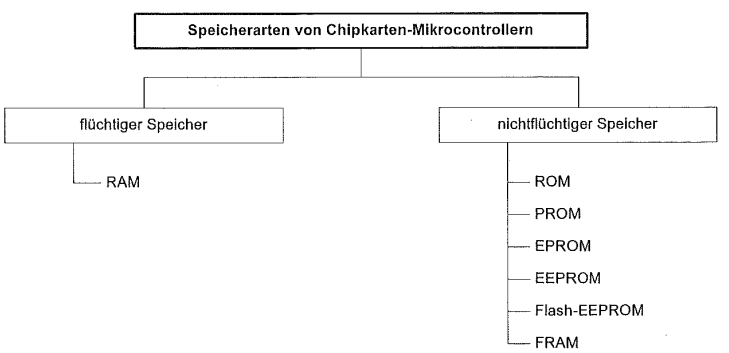


Bild 3.47 Klassifizierungsbaum der Speicherarten von Chipkarten-Mikrocontrollern. Die Speicherarten PROM, EPROM werden bei modernen Mikrocontrollern in der Regel nicht mehr verwendet. FRAM ist ein Speicher, dessen Einsatz bei Chipkarten erst am Anfang steht.

Seit einiger Zeit ist eine neue Speichertechnologie für Chipkarten von einigen Halbleiterherstellern verfügbar. Dabei handelt es sich um Flash-EEPROM-Zellen, die viel schnellere Schreib- und Löschzugriffe gestatten als die bisher üblichen EEPROM-Zellen. Die Größe beträgt je nach Realisation ungefähr die Hälfte einer heute üblichen EEPROM-Zelle.

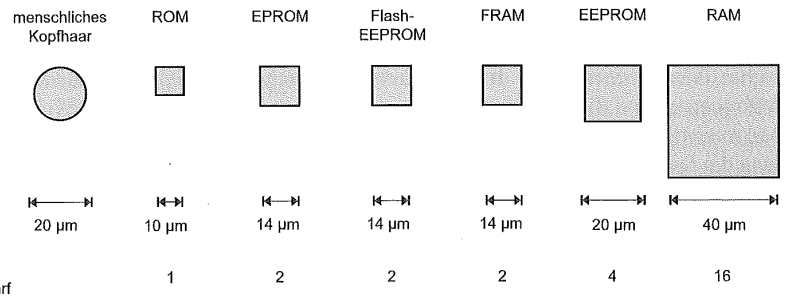


Bild 3.48 Größenvergleich des Platzbedarfs für je 1 Bit in Abhängigkeit von der Speicherart. Die Größenangaben sind circa-Werte und beziehen sich auf 0,8 µm-Technologie.
Zum Vergleich: Durchmesser des ersten Planartransistors von 1959: 764 µm [Buchmann 96, Stix 96]; Durchmesser der Punkte an den Satzenden in diesem Buch: ≈ 400 µm; Wahrnehmungsgrenze des menschlichen Auges: ≈40 µm; Größe von Bakterien: 0,4 µm – 2 µm; Durchmesser einer DNA-Doppelhelix: ≈0,1 µm.

Tabelle 3.4 In Chipkarten-Mikrocontrollern verwendete Speicherarten. Als Vergleich: Die Fläche der Punkte an den Satzenden in diesem Buch beträgt ≈125660 µm².

Speicherart	**Anzahl der Schreib-/Löschzyklen**	**Schreibzeit pro Speicherzelle**	**typische Zellgröße bei 0,8 µm-Technologie**
flüchtige Speicher			
RAM	∞	≈70 ns	≈1 700 µm²
nichtflüchtige Speicher			
EEPROM	100 000 – 1 000 000	2–10 ms	≈400 µm²
EPROM	1 (da Löschen mit UV-Licht nicht möglich)	≈50 ms	≈200 µm²
Flash-EEPROM	≈100000	≈10 µs	≈200 µm²
FRAM	$\approx 10^{10}$	≈100 ns	≈200 µm²
PROM	1	≈100 ms	---
ROM	0	---	≈100 µm²

Zur Verdeutlichung der Größenverhältnisse im Folgenden drei Zahlenbeispiele: Ein einfacher Laserdrucker arbeitet mit einer Auflösung von 600 dpi (*dots per inch*). Umgerechnet ist damit eine minimale Punktgröße von 42,6 µm möglich. Der Punkt am Ende dieses Satzes hat im Übrigen einen Durchmesser von 400 µm. Wollte man in der heute in der Halbleitertechnik noch verwendeten Strukturbreite von 0,8 µm drucken, müsste der Drucker eine Auflösung von 32000 dpi haben!

Hochleistungsfestplatten können bis zu 11,6 Milliarden Bits pro Quadratzoll speichern. Unter der idealisierten Annahme, dass ein Bit eine quadratische Fläche benötigt, ergibt sich daraus eine Kantenlänge von 0,24 µm für den notwendigen Speicherplatz eines Bit. Die ROM-Zelle eines Chipkarten-Mikrocontrollers zur Speicherung eines Bits in 0,8 µm-Technologie hat einen mehr als 1700-fachen Platzbedarf!

Anders sieht es jedoch bei einer CD-ROM aus. Dort beträgt die Speicherdichte 7,3 MByte/cm^2, was umgerechnet einer Kantenlänge von 1,3 µm für ein Bit bei quadratischer Anordnung entspricht und somit etwa 60-mal weniger Platz als eine ROM-Zelle in 0,8 µm-Technologie beansprucht. Bei DVDs (*digital versatile disc*) können bereits 50,5 MByte/cm^2 untergebracht werden. Ein Bit benötigt dort die Fläche eines Quadrats mit 0,5 µm Seitenlänge und ist somit 400-mal kleiner als eine einbittige ROM-Zelle in 0,8 µm-Technologie.

Tabelle 3.5 Typische Aufteilung der Flächen eines Chipkarten-Mikrocontrollers.

Speicherart	Flächenbedarf
CPU, NPU	20 %
ROM	10 %
EEPROM	45 %
RAM	15 %
diverses	10 %

ROM (*read only memory*)

Diese Speicherart kann nur gelesen werden, ein schreibender Zugriff ist nicht möglich, was der Name „read only memory" ja schon aussagt. Um die Daten im Speicher zu erhalten, wird keine Spannung benötigt, da die Daten durch eine feste „Verdrahtung" im Chip repräsentiert sind.

Im ROM-Speicher einer Chipkarte befinden sich die meisten Betriebssystemroutinen sowie diverse Test- und Diagnosefunktionen. Diese Programme werden vom Halbleiterhersteller bei der Produktion der Chips eingebracht. Dabei wird aus dem Programm eine so genannte ROM-Maske erstellt. Diese wird dann dazu verwendet, auf lithographischem Weg bei der Chipproduktion das Programm in den Chip einzubringen. Nur während der Halbleiterproduktion können die für alle Chips gleichen Daten für das ROM in den Chip gebracht werden.

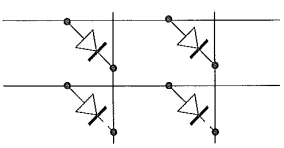


Bild 3.49 Grundsätzlicher funktioneller Aufbau eines ROMs. Bei den beiden unteren Zellen ist die Verbindung zur Diode unterbrochen.

PROM (*programmable read only memory*)

Die Speicherart PROM wird in Chipkarten-Mikrocontrollern nicht eingesetzt, obwohl sie einige Vorteile hätte. Dieser Speicher müsste im Gegensatz zum ROM nicht während der Chipproduktion programmiert werden, sondern ließe sich kurz vor dem Einbau der Chips in Module mit Daten versehen. Auch würde ein PROM zum Datenerhalt keine Spannung

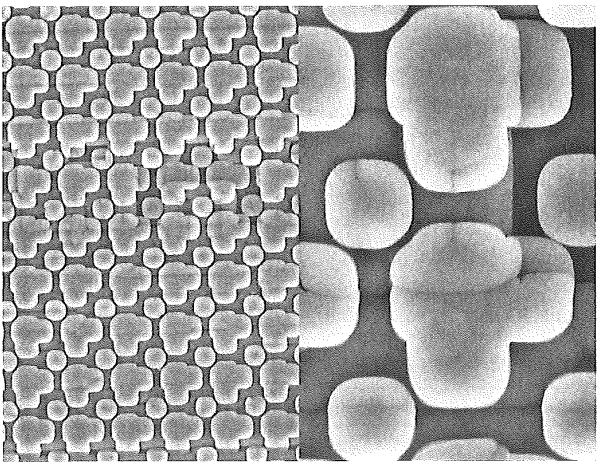


Bild 3.50 ROM-Zelle. Links in 1 000facher Vergrößerung, rechts in 11 200facher Vergrößerung. (Giesecke & Devrient)

benötigen. Der Grund, warum dieser Speichertyp keine Verwendung findet, liegt vor allem darin, dass man zur Programmierung von außen auf den Adress-, Daten- und Steuerbus zugreifen muss. Genau dies darf aber bei Chipkarten nicht möglich sein, da mit freiem Zugriff auf die Busse nicht nur Daten geschrieben, sondern auch gelesen werden können. In Anbetracht der geheimen Daten im Speicher ist dies aber strikt untersagt.

EPROM (*erasable read only memory*)
In den Anfangsjahren der Chipkartentechnik fanden oftmals EPROMs Verwendung, da es damals die einzige Speicherart war, in der auch ohne Spannung Daten erhalten blieben und (allerdings nur einmal pro Bit) geschrieben werden konnten. Da sich EPROMs aber nur mit UV-Strahlung wieder löschen lassen, konnten diese Speicher nie mehr gelöscht werden. Dies ist auch der Grund dafür, warum EPROM-Speicher heute keine praktische Bedeutung mehr haben.

Der einzige sinnvolle Einsatz ist die irreversible Speicherung einer Chipnummer bei der Halbleiterproduktion, was aber mittlerweile durch ein spezielles EEPROM ohne Löschmöglichkeit bewerkstelligt wird.

EEPROM (*electrical erasable read only memory*)
EEPROM als technisch gegenüber ROM und RAM aufwendigere Speicherart wird in Chipkarten für alle Daten und Programme verwendet, die irgendwann einmal verändert oder gelöscht werden sollen. Der Funktionalität nach entspricht ein EEPROM der Festplatte eines PCs, da Daten auch ohne Stromzufuhr erhalten bleiben und sich bei Bedarf ändern lassen. EEPROM ist also ein nichtflüchtiger Speicher.

Eine EEPROM-Zelle stellt dem Prinzip nach einen winzigen Kondensator dar, der geladen oder entladen sein kann. Der Ladezustand kann durch eine Art Sensorik abgefragt werden. Ein geladener Kondensator repräsentiert eine logische Eins und umgekehrt. Man braucht also, um ein Byte zu speichern, acht dieser kleinen Kondensatoren plus die entsprechende Sensorik.

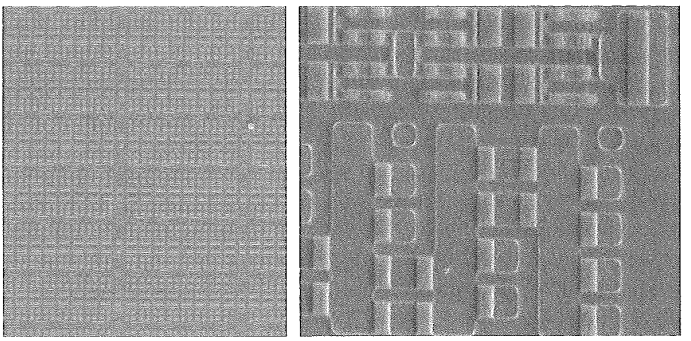


Bild 3.51 EEPROM-Zelle. Links in 1 000facher, rechts in 4 000facher Vergrößerung. (Giesecke & Devrient)

Um Daten in eine EEPROM-Zelle zu schreiben, ist vor allem der gelöschte Zustand der Zelle entscheidend, welcher bei den meisten Zellentypen °1° ist. Ein EEPROM hat nun die Eigenschaft, dass es von seinem gelöschten Zustand immer nur auf den nicht gelöschten Zustand, im Beispiel also °0°, gesetzt werden kann. Befindet sich eine EEPROM-Zelle bereits im Zustand °0°, muss die gesamte EEPROM-Seite (*page*) gelöscht werden, wenn man nur ein einziges Bit wieder in den Zustand °1° setzen möchte. Der dazu üblicherweise verwendete Algorithmus einer EEPROM-Schreibroutine wird als Pseudocode im Tabelle 3.6 aufgeführt.

Tabelle 3.6 Pseudocode einer Schreibroutine für komplette EEPROM-Pages. Sollten mehrere Pages oder nur Teile einer Page geschrieben werden, so ist diese Funktion in eine übergeordnete Programmroutine einzubetten. Analog ist zu verfahren, falls im Fehlerfall eine Schreibwiederholroutine aufgerufen werden soll. Der gelöschte Zustand des EEPROMs ist 'FF', der geschriebene somit '00'.

UpdateEEPROM: // *NewData*: zu schreibende Daten // *StoredData*: gespeicherte Daten	Einsprungstelle für das Schreiben von Daten in eine EEPROM-Page
IF (NewData = StoredData) THEN(GOTO UpdateEEPROM_Exit)	Falls die in der EEPROM-Page gespeicherten Daten bereits den neuen Daten entsprechen, dann beende die Funktion.
WorkData := NewData XOR StoredData	In der Variablen WorkData sind nun durch die XOR-Funktion die Unterschiede zwischen gespeicherten und neu zu schreibenden Daten als gesetzte Bits erkennbar.
WorkData := WorkData AND NewData	Die Variable WorkData ist nun durch die AND-Funktion ungleich null, wenn die EEPROM-Page vor dem Schreibvorgang gelöscht werden muss.

IF (WorkData <> 0) THEN(
Lösche EEPROM-Page
IF(StoredData <> 'FF') THEN _
(GOTO UpdateEEPROM_Errror_Exit))
	Falls die Variable WorkData ungleich null ist, muss die EEPROM-Page vor dem Schreibvorgang gelöscht werden. Nach dem Vorgang wird geprüft, ob die EEPROM-Page erfolgreich gelöscht werden konnte.

Schreibe EEPROM-Page mit NewData IF (StoredData <> NewData) THEN _ (GOTO UpdateEEPROM_Errror_Exit)	Die EEPROM-Page kann nun geschrieben werden. Anschließend wird geprüft, ob die Daten erfolgreich ins EEPROM geschrieben werden konnten.
UpdateEEPROM_Exit: RETURN	Die Funktion konnte erfolgreich abgeschlossen werden.
UpdateEEPROM_Error_Exit: RETURN	Bei Ausführung der Funktion ist ein Fehler aufgetreten.

Um die Funktionsweise einer EEPROM-Zelle zu verstehen, muss man sich den halbleitertechnischen Hintergrund vor Augen führen. In Bild 3.52 ist ein Schnitt einer EEPROM-Zelle gezeigt. In der Realität ist der Aufbau allerdings etwas komplizierter, doch zum Verständnis eignet sich die schematisierte Zeichnung sehr gut.

Eine EEPROM-Zelle in ihrer einfachsten Form ist grundsätzlich ein modifizierter Feldeffekttransistor (MOSFET) und wird auf einem Trägermaterial aus Silizium aufgebaut. Auf dieses Grundmaterial werden eine Source (Quelle) und ein Drain (Senke) aufgebracht. Zwischen Source und Drain befindet sich noch ein Control Gate, mit dem man durch Anlegen einer Spannung den Stromfluss zwischen Source und Drain steuern kann. Solange am Control Gate keine Spannung anliegt, ist ein Stromfluss nicht möglich, da zwei Diodenübergänge (n-p und p-n) dazwischenliegen. Legt man am Gate eine positive Spannung an, zieht dies Elektronen aus dem Substrat, und es bildet sich ein elektrisch leitfähiger Kanal zwischen Source und Drain, d. h. der FET ist leitend, und es fließt Strom.

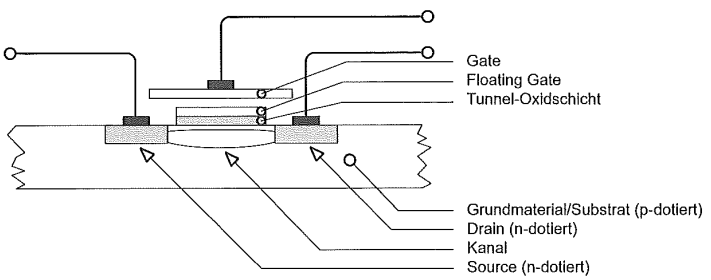


Bild 3.52 Schnittbild des halbleitertechnischen Aufbaus einer EEPROM-Zelle.

Bei einer EEPROM-Zelle liegt zwischen Control Gate und dem Grundmaterial noch ein so genanntes Floating Gate. Dieses ist mit keiner äußeren Spannungsquelle verbunden und hat einen sehr geringen Abstand in der Größenordnung von 10 nm zum Grundmaterial. Das Floating Gate kann durch den Tunnel-Effekt (Fowler-Nordheim-Effekt) über das Substrat geladen oder entladen werden. Dieser Effekt verursacht das Durchdringen von Ladungsträgern an dünnen Oxidschichten. Die Voraussetzung ist dabei eine hinreichend große Potenzialdifferenz an der dünnen isolierenden Schicht, der Tunnel-Oxidschicht. Durch die Ladung des Floating Gate kann der Stromfluss zwischen Source und Drain gesteuert werden. Dies bedeutet, dass je nach Stromfluss eine logische Null oder eine logische Eins für dieses Gatter interpretiert werden kann.

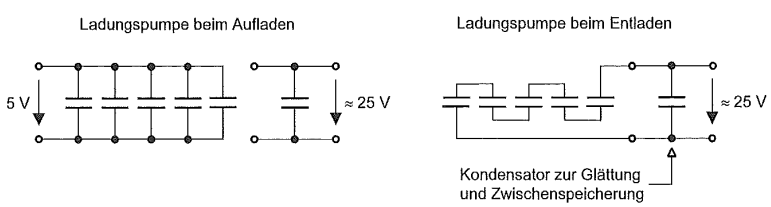


Bild 3.53 Das Prinzipschaltbild einer möglichen Schaltung für eine Ladungspumpe beim Aufladen und beim Entladen. Dieser Vorgang findet mit so hoher Frequenz statt, dass die Ladungspumpe an ihrem Ausgang einen nur noch leicht pulsierenden Gleichstrom abgibt.

Zum Laden des Floating Gates wird eine hohe positive Spannung an das Control Gate angelegt. Dies ruft eine hohe Potenzialdifferenz zwischen Substrat und Floating Gate hervor, und daraufhin tunneln Elektronen durch die Tunnel-Oxidschicht zum Floating Gate. Die dabei fließenden Ströme bewegen sich im Pico-Ampere-Bereich. Nun ist das Floating Gate negativ geladen und verursacht eine hohe Schwellenspannung zwischen Source und Drain, d. h. der Feldeffekttransistor ist sperrend. Es kann also kein Strom zwischen Source und Drain fließen. Die Speicherung von Elektronen im Floating Gate ist damit also analog der Speicherung von Information.

Die notwendige Spannung zum Laden der EEPROM-Zelle beträgt etwa 17 V am Control Gate, die sich aber durch die Kopplung auf etwa 12 V am Floating Gate reduziert. Da Chipkarten-Mikrocontroller aber nur mit 1,8 V bis 5 V versorgt werden, benötigt man eine Ladungspumpe, die dem Prinzip nach eine kaskadierte Spannungsverdopplerschaltung ist. Diese erzeugt aus der niedrigen Eingangsspannung eine Ausgangsspannung von etwa 25 V, die sich nach Stabilisierung ungefähr in der Höhe der benötigten 17 V bewegt. Der Vorgang des Ladens einer EEPROM-Zelle benötigt je nach Aufbau für eine Speicherseite (d. h. 1 Byte bis 32 Byte) zwischen 2 ms und 10 ms.

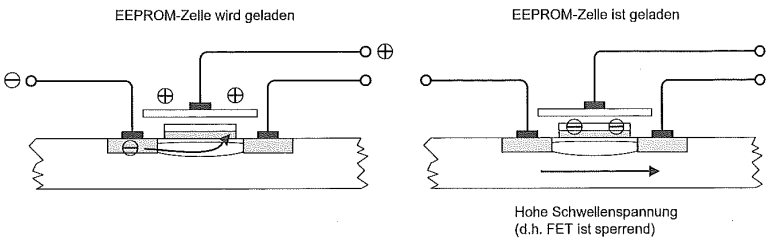


Bild 3.54 Das Laden einer EEPROM-Zelle.

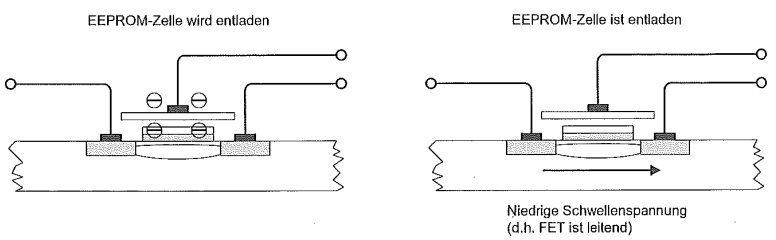


Bild 3.55 Das Entladen einer EEPROM-Zelle.

Zum Löschen der EEPROM-Zelle legt man eine Spannung negativer Polarität an das Control Gate, und die Elektronen bewegen sich aus dem Floating Gate zurück in das Substrat. Die EEPROM-Zelle ist nun entladen und die Schwellenspannung zwischen Source und Drain niedrig, d. h. der FET ist leitend.

Durch Erhitzen oder durch starke Strahlung (z. B. Röntgen, UV) kann das Floating Gate ebenfalls entladen werden, d.h. es nimmt den so genannten sicheren Zustand ein. Dieser Zustand ist für das Design von Chipkarten-Betriebssystemen von elementarer Bedeutung,

da sonst Sicherheitsbarrieren durch vorsätzliche Änderung von Umweltbedingungen durchbrochen werden können. Je nach technischer Realisierung einer EEPROM-Zelle kann der sichere Zustand dem logischen Wert null oder eins entsprechen. Dies ist für jeden Chipkarten-Mikrocontroller spezifisch und muss bei Bedarf vom Hersteller erfragt werden.

Das EEPROM ist einer der wenigen Halbleiterspeicher, die eine begrenzte Anzahl von Zugriffen haben. Gelesen werden kann dieser Speicher unbegrenzt oft, doch ist die Anzahl der Programmierungen limitiert. Der Grund für diese Begrenzung ist im halbleitertechnischen Aufbau des Speichers zu suchen. Die Lebensdauer hängt stark von Art, Dicke und Qualität der Tunnel-Oxidschicht zwischen Floating Gate und Substrat ab. Da diese Schicht schon in einer sehr frühen Phase beim Aufbau des Halbleiters erzeugt werden muss, ist sie natürlich in den weiteren Produktionsschritten großen thermischen Belastungen ausgesetzt. Diese Belastungen können eine Schädigung der Tunnel-Oxidschicht hervorrufen, was sich auf die Lebensdauer der EEPROM-Zelle auswirkt. Während der halbleitertechnischen Fertigung und bei jedem Schreiben einer EEPROM-Zelle fängt die Tunnel-Oxidschicht Elektronen ein, die sie nicht wieder abgibt. Diese eingefangenen Elektronen befinden sich näher am Kanal zwischen Source und Drain und können sich ab einer bestimmten Anzahl stärker auf die Schwellenspannung auswirken als die Ladung auf dem Floating Gate. Ist dies der Fall, hat die EEPROM-Zelle das Ende ihrer Lebensdauer erreicht. Sie lässt sich zwar noch schreiben, doch beeinflusst die Ladung auf dem Floating Gate nur mehr unwesentlich die Kanaleigenschaften zwischen Source und Drain, was dazu führt, dass die Schwellenspannung immer den gleichen Wert beibehält. Die Anzahl der insofern möglichen Schreib-/Löschzyklen kann durch den halbleitertechnischen Aufbau sehr unterschiedlich sein. Übliche Werte bewegen sich zwischen 100000 und 1000000 Zyklen über den gesamten Temperatur- und Versorgungsspannungsbereich. Bei optimaler Spannungsversorgung und Raumtemperatur kann die Anzahl der möglichen Zyklen jedoch 10- bis 50fach höher sein.

Befindet sich eine EEPROM-Zelle an der Grenze ihrer Lebensdauer, werden Informationen nur noch über eine kurze Zeit gespeichert. Diese Zeitspanne kann sich im Bereich von Stunden bis zu Minuten oder Sekunden bewegen. Je „verbrauchter" das EEPROM ist, d. h. je mehr Elektronen die Tunnel-Oxidschicht aufgenommen hat, desto kürzer ist die Speicherdauer.

Ein geladenes Floating Gate verliert aufgrund von Isolationsverlusten und quantenmechanischen Effekten mit der Zeit seine Ladung. Die Zeitdauer, ab der sich dies bemerkbar macht, kann zwischen 10 und 100 Jahren liegen. Interessant ist in diesem Zusammenhang

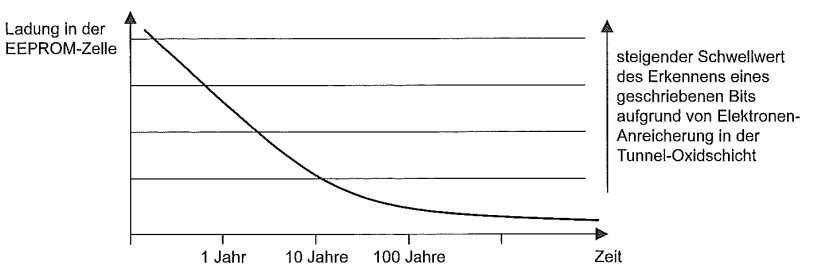


Bild 3.56 Verschiebung der Entladekurve einer EEPROM-Zelle in Abhängigkeit der durchgeführten Anzahl der Programm-/Löschzyklen.

noch, dass ein geladenes Floating Gate je nach technologischer Ausführung zwischen 100000 und 1000000 Elektronen enthält. Momentan garantieren alle Halbleiterhersteller 10 Jahre für den sicheren Datenerhalt. Um dies zu erhöhen, kann der Inhalt der EEPROM-Zellen zyklisch durch eine Nachprogrammierung wieder aufgefrischt werden. Dies ergibt aber nur dann Sinn, wenn Daten über lange Zeit aufbewahrt werden sollen.

Flash-EEPROM (***flash electrical erasable read only memory***)
Flash-EEPROM-Speicher, oft kurz Flash-Speicher genannt, ist analog dem EEPROM-Speicher nichtflüchtig, d.h. auch nach Abschalten der Versorgungsspannung bleibt der Dateninhalt erhalten. Sie ähneln in Funktionalität und halbleitertechnischem Aufbau stark den EEPROM-Zellen. Der grundlegende Unterschied zum EEPROM ist der Schreibvorgang, der nicht durch den Fowler-Nordheim-Effekt, sondern durch eine so genannte Hot-Electron-Injection stattfindet.

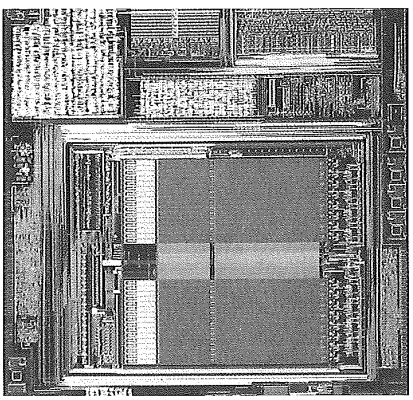


Bild 3.57 Chipkarten-Mikrocontroller AT89SC168 mit Flash-EEPROM. Die Funktionselemente in der oberen Reihe sind von links nach rechts: Logik-Baugruppe, RAM und CPU. Die am unteren Rand gelegenen Funktionselemente sind von links nach rechts: Ladungspumpe für EEPROM und Flash-EEPROM. (ATMEL)

Bei der Hot-Electron-Injection werden durch eine hohe Potenzialdifferenz zwischen Source und Drain schnelle Elektronen erzeugt, von denen, durch ein positiv geladenes Control Gate verursacht, ein Teil die Tunnel-Oxidschicht durchdringt und im Floating Gate gespeichert wird. Durch diesen Effekt reduziert sich die Schreibzeit auf ca. 10 µs, was einen großen Fortschritt gegenüber den 2–10 ms der bisherigen EEPROM-Zellen darstellt. Die äußerst kurze Schreibzeit ist im Übrigen auch der Namensgeber der Flash-Speicher. Als weiterer Vorteil ist zu nennen, dass die notwendige Programmierspannung nur etwa 12 V gegenüber 17 V bei EEPROM-Zellen beträgt.

Es gibt einige Chipkarten-Mikrocontroller mit Flash-EEPROM auf dem Markt, wobei diese Speicherart vor allem als Ersatz für das maskenprogrammierte ROM benutzt wird. Durch die nicht mehr notwendige ROM-Maske bei Mikrocontrollern mit Flash-EEPROM reduziert sich die Durchlaufzeit bei einem Chipkarten-Projekt um mehrere Monate.

Leider ist es halbleitertechnologisch extrem schwierig, EEPROM- und Flash-EEPROM-Zellen gleichzeitig auf einem Chip aufzubauen, weshalb man in der Praxis auf typischen

Mikrocontrollern mit Flash-Speicher demzufolge kein EEPROM findet. Dieses wird durch ein Flash-EEPROM mit möglichst geringer Page-Größe in der Größenordnung von 8 Byte substituiert, sodass sich für das Chipkarten-Betriebssystem kein allzu großer Unterschied zu dem üblichen EEPROM ergibt. Die Page-Größe des Flash-Speichers für den ROM-Ersatz wird in aller Regel deutlich größer festgelegt (z. B. 64–128 Byte), da die in diesem Speicher abgelegten Programmblöcke selten geschrieben werden. Bei der Halbleiterfertigung wird dabei in ein ROM kleiner Größe ein Bootloader eingebracht, der dann beim Chipkartenhersteller zum Nachladen von Programmcode und Daten benutzt wird.

Die heutigen Flash-EEPROM-Zellen besitzen einen garantierten Datenerhalt von mindestens 10 Jahren, mindestens 100000 Schreib-/Löschzyklen und eine Page-Größe von typischerweise 8 Byte bis 128 Byte.

Vereinzelt existieren Chipkarten-Mikrocontroller mit unüblich großem Speicher, oft in der Größenordnung von 1–2 MByte. Diese werden immer auf der Grundlage von Flash-Speichern mit zum Teil 64 kByte großer Page-Größe gefertigt. Dadurch wird erheblich Platz für Adress- und Steuerleitungen eingespart, sodass sich diese Speichergrößen auf einen Chip der maximal möglichen Größe von 25 mm^2 verwirklichen lassen.

FRAM (*ferroelectric random access memory*)

FRAM ist trotz seines missverständlichen Namens kein flüchtiger Speicher wie das RAM, sondern es behält seinen Inhalt auch ohne Versorgungsspannung bei. Für diese Speicherart benutzt man zur Speicherung von Informationen die Eigenschaften von ferroelektrischen Substanzen. Der Aufbau ähnelt dem der EEPROM-Zellen, nur befindet sich zwischen Control Gate und Floating Gate ein Ferroelektrikum.

Für Chipkarten wäre diese Speicherart ideal, da sie alle gewünschten Eigenschaften eines optimalen Speichers aufweist. Man benötigt zur Programmierung nur 5 V, die Programmierzeit bewegt sich im Bereich von 100 ns, und die Zahl der möglichen Programmierzyklen ist im Bereich einer Billion. Die Integrationsdichte ähnelt dem der Flash-EEPROMs. Allerdings haben FRAM-Speicher zwei Nachteile. Der Erste besteht darin, dass die Anzahl der Lesezyklen limitiert ist, was deshalb eine Art Refresh-Zyklus erfordert. Der zweite Nachteil ist allerdings von größerer Bedeutung: Die Herstellung beinhaltet

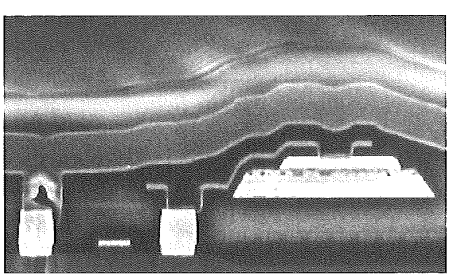


Bild 3.58 Querschnitt einer FRAM-Zelle in 0,5 µm-Technologie. Die beiden hellen Rechtecke auf der linken Seite sind die Ansteuerleitungen auf dem Halbleiter. Das auf der rechten Seite waagrecht liegende Rechteck ist die eigentliche FRAM-Zelle, mit dem Ferroelektrikum als dunkel erscheinende Schicht zwischen den beiden Elektroden. Die eigentliche FRAM-Zelle hat eine Breite von ca. 2 µm. (Fujitsu)

schwer beherrschbare Prozessschritte und es gibt im Bereich der Chipkarten-Mikrocontroller bisher wenig Bestrebungen, diese neue Technologie einzusetzen. Dies kann sich jedoch in einigen Jahren ändern, dann hätte die FRAM-Technik alle notwendigen Eigenschaften, um die heute fast ausschließlich verwendeten EEPROMs vollständig abzulösen.

RAM (*random access memory*)
Das RAM ist der Speicher einer Chipkarte, in dem Daten während einer Sitzung gespeichert und geändert werden können. Die Anzahl der möglichen Zugriffe ist unbegrenzt. Zur Funktionsfähigkeit benötigt es eine Spannungsversorgung. Ist die Betriebsspannung nicht mehr vorhanden oder fällt sie kurzzeitig aus, ist der Inhalt des RAMs nicht mehr definiert.

Aufgebaut ist eine RAM-Zelle aus mehreren Transistoren, die so geschaltet sind, dass sie als bistabile Kippschaltung funktionieren. Der Schaltungszustand repräsentiert dann den Speicherinhalt eines Bits im RAM. Das bei Chipkarten verwendete RAM ist statisch (SRAM), d. h. der Speicherinhalt muss nicht periodisch aufgefrischt werden. Damit ist es auch unabhängig von einem äußeren Takt, im Gegensatz zu einem dynamischen RAM (DRAM). Statische RAMs sind deshalb wichtig, weil es möglich sein muss, die Taktversorgung von Chipkarten anzuhalten, was bei dynamischen RAMs zum Verlust der gespeicherten Information führen würde.

3.4.3 Zusatzhardware

Ausgehend von der Hardware gebräuchlicher Mikrocontroller existieren bei Chipkarten spezielle Anforderungen, die nicht durch Software gelöst werden können und deshalb durch zusätzliche Hardwarekomponenten abgedeckt werden müssen. Aus diesem Grund bieten die verschiedenen Hersteller von Chipkarten-Mikrocontrollern eine große Bandbreite von Zusatzfunktionen als Hardware auf dem Chip an.

Im Folgenden sind die gebräuchlichsten Komponenten für Zusatzfunktionen aufgeführt. Sie müssen jedoch nicht alle gleichzeitig Bestandteil eines Mikrocontrollers sein. Dies hängt unter anderem sehr stark von der jeweiligen Zielanwendung ab. Beispielsweise wäre es aus wirtschaftlichen Gesichtspunkten unsinnig, einen RSA-Koprozessor auf einem Mikrocontroller zu integrieren, dessen Zielanwendung lediglich symmetrische Kryptoalgorithmen verwendet. Vereinzelt sind jedoch auch Mikrocontroller auf dem Markt, auf denen beinahe alle nachfolgend aufgeführten Komponenten verwirklicht sind.

Eine weitere Facette hinsichtlich Zusatzfunktionalität von Chipkarten-Mikrocontrollern findet sich rund um das große Thema Sicherheit. Im entsprechenden Abschnitt des Kapitels 8 – Sicherheitstechnik – gibt es eine umfangreiche Aufstellung von in Hardware verwirklichten Zusatzfunktionalitäten, die vor allem dazu dienen, etwaigen Angriffen zu begegnen. Die folgende Aufstellung umfasst deshalb nur Komponenten, deren unmittelbarer Fokus nicht die Verbesserung der Angriffssicherheit ist.

Hardware-unterstützte Datenübertragung, UART
Die einzige Kommunikationsmöglichkeit einer Chipkarte mit der Außenwelt ist die Datenübertragung über eine bidirektionale serielle Schnittstelle. Empfang und Senden von Daten über diese Schnittstelle wurden ursprünglich nur von der Software des Betriebssystems ohne jegliche Hardwareunterstützung gesteuert. Der programmtechnische Aufwand dafür ist sehr

hoch. Ebenso ergeben sich dadurch zusätzliche Fehlermöglichkeiten in der Software. Das Hauptproblem dabei ist jedoch die Tatsache, dass die Datenübertragung nur bis zu einer bestimmten Geschwindigkeit mit Software machbar ist, da die Prozessorgeschwindigkeit hier Grenzen setzt. Die Untergrenze bei heutigen Prozessoren liegt im Bereich des Teilers 30 (*clock rate conversion factor*), was bei 3,5 MHz zu einer Datenübertragungsrate von etwa 115 kBit/s führt.

Möchte oder muss man mit höheren Übertragungsgeschwindigkeiten kommunizieren, benötigt man entweder eine interne Taktvervielfachung oder einen UART-Baustein. Die Abkürzung UART steht für *universal asynchronous receiver transmitter* und bezeichnet einen universell einsetzbaren asynchronen Baustein zum Senden und Empfangen von Daten, unabhängig vom Prozessor. Damit ist man nicht an die Geschwindigkeit des Prozessors gebunden und benötigt auch keine Software für die Kommunikation auf Byte-Ebene. Die höheren Protokollschichten der jeweiligen Übertragungsprotokolle müssen natürlich weiterhin als Software in der Chipkarte vorhanden sein, doch die unterste Schicht kann mit dem UART-Baustein in Hardware realisiert werden.

Heutige UARTs beherrschen meist alle typischen Teiler nach ISO/IEC 7816-3 kleiner als 372 und können zum Teil bis zum Teiler 1 senden und empfangen. Bei der zur Verfügung gestellten Funktionalität gibt es eine große Bandbreite unterschiedlicher Ausführungen. Manche UART-Bausteine können nur ein einzelnes Byte senden und empfangen und beherrschen manchmal auch noch im Übertragungsfehlerfall die Byte-Sendewiederholung nach dem T = 0 Protokoll. Der Prozessor muss beim Senden und Empfangen dann lediglich zeitgerecht den UART mit den entsprechenden Daten versorgen bzw. entsorgen. Der vollständige Empfang eines Bytes kann vom UART beispielsweise durch ein Flag oder einen Interrupt dem Prozessor mitgeteilt werden. Höherentwickelte UARTs können aber auch durchaus mehrere übergebene Bytes hintereinander empfangen oder senden. Das technische Maximum an Funktionalität bieten derzeit UARTs, welche ohne Zutun und parallel zum Prozessor per DMA (*direct memory access*) direkt Sequenzen von Bytes aus dem RAM senden bzw. dort nach Empfang ablegen können.

Die technische Machbarkeit von UARTs auf Chipkarten-Mikrocontrollern war im Übrigen schon seit den Ursprüngen der Chipkarten gegeben, doch haben bis Ende der 90er-Jahre Sende- und Empfangsroutinen realisiert als Software im ROM weniger physikalische Speicherfläche auf dem Silizium benötigt als eine in der Funktion vergleichbare UART-Komponente auf dem Chip. Da der Verbrauch an Chipfläche aber entscheidend für den Preis von Chipkarten-Mikrocontrollern ist, hatten sich lange Zeit fast alle Halbleiterhersteller gegen diese Hardware entschieden. Mit steigender Integrationsdichte der Schaltungen haben sich aber die Voraussetzungen dafür geändert. Mittlerweile ist ein UART mit durchaus unterschiedlicher Funktionalität bei allen Mikrocontrollern für Chipkarten eine Standardkomponente.

Viele der neueren Mikrocontroller erlauben optional als Komponente auf dem Chip zusätzlich zum UART einen USB-Schnittstellenbaustein. Mit diesem wäre es Hardware-unterstützt möglich, via USB-Protokoll Daten zum Terminal zu senden und von diesem auch zu empfangen. Leider ist die Benutzung dieser USB-Hardware-Erweiterung bislang *de facto* nicht möglich, da USB auf Chipkarten noch durch keine Norm abgedeckt wird und somit keine Kompatibilität der Chipkarten untereinander und zu Terminals gewährleistet ist.

Timer, Watchdog

Ein Timer auf einem Chipkarten-Mikrocontroller ist über einen einstellbaren Teiler mit dem internen Prozessortakt oder UART-Takt (der die etus zählt) verbunden und hat in der Regel einen Zählbereich von 16 bit oder seltener 32 bit. Mit einem Timer ist es möglich, die Anzahl der Takte vom Startbefehl bis zum Endebefehl zu messen, ohne dazu den Prozessor in Anspruch zu nehmen. Die meisten Timer ermöglichen auch ein zyklisch wiederholtes Herabzählen von einem vorgegebenen Wert mit Auslösen eines Interrupts und automatisches Rücksetzen des Timers auf den Ausgangswert, wenn der Wert Null erreicht ist.

Oft befindet sich auf dem Mikrocontroller auch ein Watchdog. Dies ist im Prinzip ein Timer, der regelmäßig per explizitem Befehl des Prozessors zurückgesetzt werden muss, damit er nicht nach einer einstellbaren Zeitspanne einen Reset auslöst. Damit kann der Mikrocontroller automatisch und nach einer festlegbaren maximalen Zeitspanne wieder auf einen definierten Zustand zurückgesetzt werden, falls er in eine Endlosschleife eingetreten sein sollte. Watchdogs werden typischerweise vor allem im typischen autonomen Controller-Umfeld eingesetzt und leisten dort auch hervorragende Dienste. Bei Chipkarten haben Watchdogs jedoch wenig Sinn, da zum einen die Software (hoffentlich) extrem zuverlässig arbeitet und im Falle einer Endlosschleife das Terminal durchaus die Möglichkeit hat, diese durch einen Reset abzubrechen. Deshalb werden die Watchdog-Timer auf den Chipkarten-Mikrocontrollern bisher in aller Regel nicht verwendet.

Interne Taktvervielfachung/-erzeugung

Von Chipkarten werden immer höhere Rechenleistungen gefordert. Dies betrifft sowohl den Prozessor als auch den Bereich für kryptografische Algorithmen. Um diesen Forderungen nachzukommen, könnte man einfach eine höhere Taktfrequenz an den Mikrocontrollern anlegen. Die Rechenleistung stiege dann direkt proportional mit dem angelegten Takt. Eine Verdoppelung der Taktfrequenz würde damit auch eine Verdoppelung der Rechenleistung des Prozessors zur Folge haben. Leider ist es aber aus Kompatibilitätsgründen zu den einschlägigen Normen im Allgemeinen nicht möglich, die Taktfrequenz über 5 MHz hinaus zu erhöhen.

Um diese Beschränkung zu umgehen, kann die intern für die Komponenten des Mikrocontrollers verfügbare Taktfrequenz mit einer auf dem Chip enthaltenen Taktvervielfachung erhöht werden. Technisch realisiert wird dies mit einer PLL-Schaltung (*phase locked loop*), welche seit langem Stand der Technik ist, oder RC-Oszillatoren. So kann beispielsweise eine Chipkarte mit einem außen angelegten Takt von 5 MHz intern mit 20 MHz betrieben werden. Für die Berechnungszeit von komplexen kryptografischen Algorithmen oder den Betrieb einer Java Virtual Maschine bringt dies erhebliche Vorteile.

Allerdings darf man bei der internen Taktvervielfachung bzw. Takterzeugung nicht übersehen, dass der Stromverbrauch entsprechend ansteigt. In der Regel herrschen hier lineare Zusammenhänge, d. h. dass beispielsweise die Vervierfachung des Taktes eine Vervierfachung des Stromverbrauchs hervorruft. Gerade bei batteriebetriebenen Terminals ist aber ein hoher Stromverbrauch unerwünscht.

Eine elegante Lösung dieses Problems ist ein so genanntes intelligentes Leistungsmanagement auf dem Mikrocontroller. Dabei wird der Steuerlogik des PLL-Oszillators der maximal erlaubte Stromverbrauch des Mikrocontrollers mitgeteilt. Diese regelt dann ohne

weiteres Zutun des Prozessors den PLL-Oszillator immer in einen Frequenzbereich, sodass der Stromverbrauch den vorgegebenen Maximalwert nicht überschreitet. Wird beispielsweise innerhalb des Regelkreises die erheblichen Strom benötigende NPU zugeschaltet, so wird automatisch die interne Taktfrequenz gesenkt, sodass der Stromverbrauch nicht über den vorgegebenen Wert steigt. Leider hat diese Lösung einen kleinen Wermutstropfen. Die Spezifikationen für GSM- und UMTS-Telekommunikations-Chipkarten verbieten (zurzeit) die Benutzung von freischwingenden Oszillatoren auf Chipkarten-Mikrocontrollern. Begründet wird dies mit möglichen Interferenz-Effekten mit der restlichen Elektronik der Mobiltelefone. Solange diese Teile der Spezifikationen Bestand haben, können deshalb weder eine stufenlose Anpassung der internen Taktfrequenz noch vollständig vom angelegten Takt unabhängige Oszillatoren benutzt werden. Diese Spezifikationen erlauben im Übrigen jedoch sehr wohl eine Taktfrequenzvervielfachung, solange externer und interner Takt miteinander über einen festen Multiplikator miteinander gekoppelt sind.

Jedoch ist die Prozessorgeschwindigkeit nicht der einzige Flaschenhals einer Chipkarte. Die Geschwindigkeit der von den Normen festgelegten Datenübertragung und die Schreib-/Löschzeit des EEPROMs profitieren nicht von einer solchen Lösung, was dann doch die Vorteile etwas beschränkt. Für manche Anwendungen kann es von großem Vorteil sein, eine intern höher getaktete Chipkarte zu verwenden, vor allem, wenn man in Betracht zieht, dass der Schaltungsaufwand und somit der Platzbedarf auf dem Chip gering sind. Aus diesem Grund ist die Möglichkeit der internen Taktvervielfachung bei nahezu allen neuen Chipkarten-Mikrocontrollern gegeben.

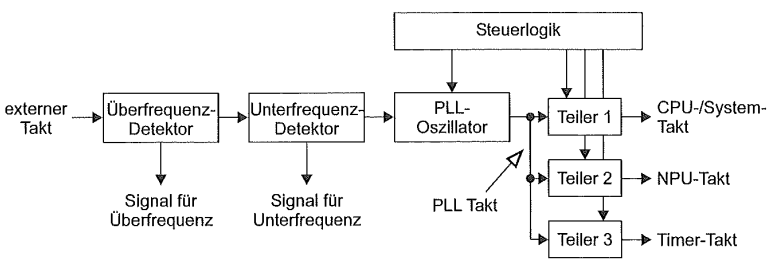


Bild 3.59 Blockschaltbild einer möglichen internen Taktvervielfachung mit PLL-Oszillator und nachgeschalteten Teilern für die verschiedenen Komponenten auf einem Mikrocontroller. Der Takt der NPU wird oftmals direkt und ohne vorgeschalteten Teiler dem PLL entnommen. Sind mehrere Timer auf dem Mikrocontroller vorhanden, so besitzen diese in der Regel auch jeweils separat einstellbare vorgeschaltete Teiler.

DMA (*direct memory access*)

DMA-Einheiten sind im PC-Bereich seit langer Zeit eingeführte Komponenten. Mittels DMA können unabhängig und zum Teil auch parallel zum Prozessor in hoher Geschwindigkeit Daten zwischen zwei oder mehreren Speicherbereichen kopiert oder verschoben werden. Oftmals ist auch das selbstständige Füllen eines bestimmten Speicherbereichs mit einem vorgegebenen Wert möglich. Der Haupteffekt einer DMA-Einheit besteht darin, den Prozessor zu entlasten und insofern bestimmte Programmteile einfacher zu gestalten. Bislang sind leistungsfähige DMA-Komponenten auf Chipkarten-Mikrocontrollern allerdings erst in Ansätzen vorhanden.

Hardware-unterstützte Speicherverwaltung, Firewall, MMU (*memory management unit*)

Bei neueren Chipkarten-Betriebssystemen ist es möglich, ausführbaren Maschinencode in die Karte nachzuladen.[4] Dieser kann dann mit einem speziellen Befehl aufgerufen werden und beispielsweise eine nur dem Kartenherausgeber bekannte kryptografische Funktion ausführen. Sobald ein ausführbares Programm in die Karte geladen wird, kann man aber prinzipiell nicht mehr verhindern, dass dessen Funktion unter anderem auch das Lesen von geheimen Speicherinhalten beinhalten kann. Nun achten aber die Betriebssystem-Hersteller sehr darauf, dass der Aufbau und der Programmcode ihrer Betriebssysteme geheim bleibt. Ähnlich verhält es sich mit geheimen Schlüsseln oder Algorithmen innerhalb von verschiedenen Anwendungen auf einer Karte. Für einen Anwendungsanbieter hätte ein Bekanntwerden seiner auf der Karte befindlichen Geheimnisse fatale Folgen. Eine verwaltungstechnische Lösung dieses Problems ist die Prüfung des einzubringenden Programms durch eine unabhängige Instanz. Aber auch dies kann keine vollständige Sicherheit gewährleisten, da man später ein anderes Programm in die Karte laden könnte als das geprüfte oder der Programmcode so geheim sein muss, dass ihn niemand außer dem Anwendungsanbieter kennen darf.

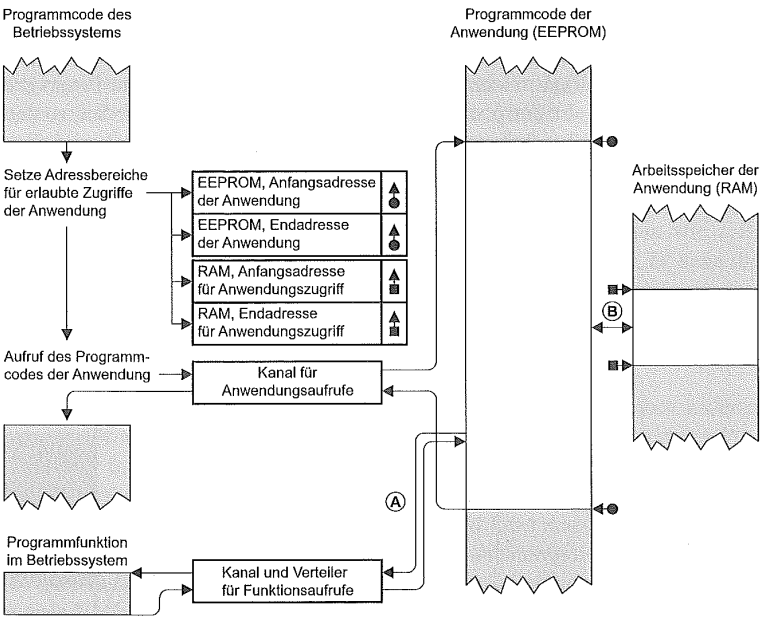


Bild 3.60 Schematische Darstellung der Funktionsweise einer Hardware-unterstützten Speicherverwaltung (MMU) auf einem Chipkarten-Mikrocontroller. Der Ablauf bei A zeigt einen durch die MMU kanalisierten und durch einen Verteiler gesteuerten Aufruf einer Funktion des Chipkarten-Betriebssystems. Bei B ist beispielhaft ein Schreib-Lesezugriff auf den durch die MMU begrenzten Arbeitsspeicher der Anwendung eingezeichnet.

[4] Siehe auch Abschnitt 5.10 „Chipkarten-Betriebssysteme mit nachladbarem Programmcode".

Um nun einen akzeptablen Ausweg zu finden, kann man einen Chipkarten-Mikrocontroller mit einer so genannten Memory Management Unit, kurz MMU, ausrüsten. Diese kontrolliert parallel zur Programmausführung die Speichergrenzen der aktuellen Anwendung. Der erlaubte Speicherbereich wird vor dem Aufruf durch eine Betriebssystemroutine festgelegt und kann dann während des Ablaufs des Programms in der Anwendung nicht mehr verändert werden. Damit ist sichergestellt, dass eine Anwendung vollständig in sich gekapselt ist und nicht auf für sie verbotene Speicherbereiche zugreifen kann. Die damit geschaffenen Trennbereiche bezeichnet man in Anlehnung an den Brandschutz als Firewalls. Wird aus dem durch Firewalls eingegrenzten Speicherbereich heraus versucht, auf andere Speicherbereiche zuzugreifen, dann ist dies zum einen nicht möglich und zum anderen wird in aller Regel ein Interrupt ausgelöst, da diese Verletzung sofort detektiert werden kann.

Nur sehr wenige Mikrocontroller für Chipkarten besitzen momentan die in vielen Bereichen schon seit Jahren eingesetzten MMUs. Doch wird diese Zusatzhardware in Zukunft stark an Bedeutung gewinnen, da es der einzige gangbare Weg ist, mehrere Anwendungen innerhalb einer Chipkarte in Hardware sicher voneinander zu separieren.

Ein weiterer Gesichtspunkt von MMUs ist die Funktion, physikalisch adressierbaren Speicher auf beliebige Adressen innerhalb des logischen Adressraum des Prozessors zu legen. Dies vereinfacht zum Teil erheblich die Speicherverwaltung des Chipkarten-Betriebssystems und ermöglicht ebenfalls eine striktere Trennung von Anwendungen hinsichtlich des Adressraums. Zudem kann im Falle von nachladbarem nativen Programmcode dieser über die MMU an die entsprechende Speicheradresse relokiert werden, so das eine manuelle Relokation der Programmcodes über das Betriebssystem unnötig wird.

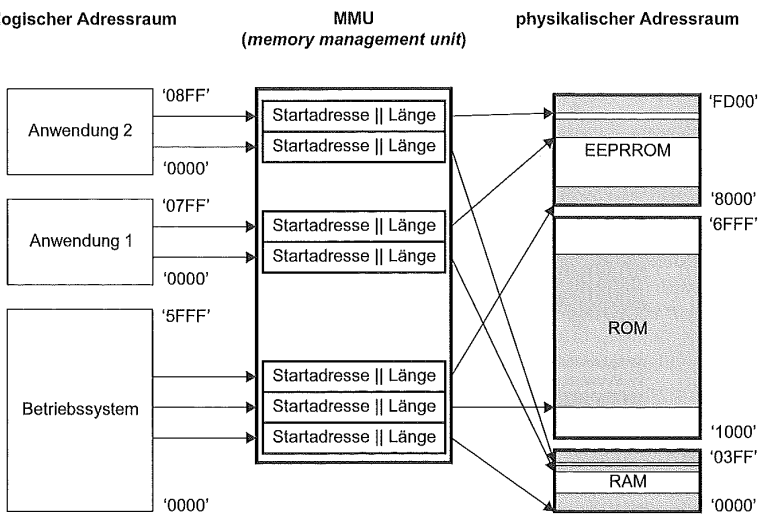


Bild 3.61 Schematische Darstellung der Arbeitsweise einer Hardware-unterstützten Speicherverwaltung (MMU) auf einem Chipkarten-Mikrocontroller hinsichtlich der Zuordnung von logischem zu physikalischem Adressraum. Das Beispiel zeigt ein Betriebssystem und zwei Anwendungen, die sich über die MMU den physikalisch vorhandenen Speicher teilen, welcher für die drei Zugriffsinstanzen von der MMU logisch jeweils in einen Adressbereich ab '0000' übersetzt wird.

Bei der Benutzung von MMUs in Chipkarten-Betriebssystemen muss allerdings ein kritischer Punkt beachtet werden. Beim aktuellen Stand der Technik sind alle MMUs auf den jeweiligen Mikrocontroller maßgeschneidert. Damit werden zwar Funktion und Flächenbedarf der MMU optimiert, allerdings auf Kosten der Portabilität des Programmcodes. Die Verwendung einer bestimmten MMU hat in der Praxis deutlich größere Auswirkungen auf das Betriebssystem als der jeweils verwendete Prozessor. Bei Chipkarten-Betriebssystemen für Massenanwendungen, bei denen der Zwang zur Unterstützung von mehreren Hardwareplattformen besteht, wurden deshalb bisher MMUs nur sehr verhalten eingesetzt.

CRC-Recheneinheit (*cyclic redundancy check calculation unit*)
Die Berechnung eines CRC-Codes ist immer noch eine häufig verwendete Methode, um Daten oder Programme mit einer Fehlererkennung abzusichern. Die CRC-Berechnung ist durch die intensiv notwendigen Bitoperationen in Software verhältnismäßig langsam und kann auch sehr gut als Hardware-Komponente auf dem Mikrocontroller in Silizium umgesetzt werden. Es gibt Mikrocontroller für Chipkarten, die aus diesem Grund eine hardwarebasierte CRC-Recheneinheit haben. Selbstverständlich müssen dabei die die üblichen Generatorpolynome und Startwerte einstellbar sein.

Zufallszahlengenerator (*random number generator – RNG*)
Zur Schlüsselgenerierung in der Chipkarte und zur Authentisierung von Chipkarte und Terminal werden oft Zufallszahlen benötigt, die aus Gründen der Sicherheit echte Zufallszahlen sein sollten und nicht pseudozufällig, wie bei typischen Software-basierten Zufallszahlengeneratoren üblich. Alle neuen Chipkarten-Mikrocontroller haben einen Zufallszahlengenerator in Hardware, der echte Zufallszahlen liefert.

Dieser Generator darf aber nicht durch äußere physikalische Eigenschaften wie z. B. Temperatur oder Versorgungsspannung in seiner Güte beeinträchtigt werden. Er kann die Zufallszahlen unter Zuhilfenahme dieser äußeren Einwirkungen auf dem Chip erzeugen, doch muss dies in einer Art und Weise geschehen, dass durch gezielte Veränderung eines oder mehrerer dieser Parameter keine Vorhersagbarkeit der erzeugten Zufallszahlen möglich ist.

Da sich dies sehr schwierig in Silizium implementieren lässt, geht man einen anderen Weg. Die Zufallszahlengeneratoren verwenden verschiedene logische Zustände des Prozessors wie Takt oder Speicherinhalte und geben diese auf ein rückgekoppeltes Schieberegister (LFSR – *linear feedback shift register*), das mit einem gleichfalls aus verschiedenen Parametern erzeugten Takt weitergeschaltet wird. Dieser Takt kann manchmal ein Mehrfaches des Prozessortaktes sein. Liest nun die CPU das Zufallszahlenregister aus, dann erhält sie eine relativ gute Zufallszahl, die von außen nicht deterministisch ermittelbar ist. Durch darauf aufbauende Verfahren und Algorithmen lässt sich die Güte der so erhaltenen echten Zufallszahl weiter verbessern. Wichtig ist aber in diesem Zusammenhang, dass bereits derhardwarebasierte Zufallszahlen-Generator gute Zufallszahlen liefert, die auch die üblichen Tests (z. B. FIPS 140-2) bestehen.[5]

[5] Ein Überblick zum Thema Güte von Zufallszahlen befindet sich in Abschnitt 4.10.2 „Prüfung von Zufallszahlen".

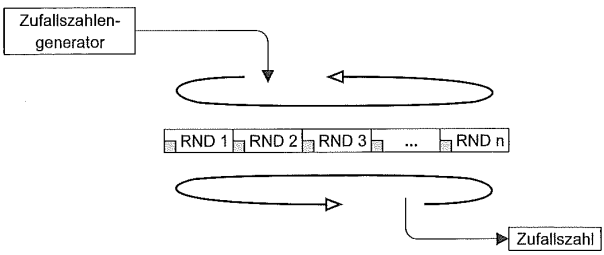


Bild 3.62 Beispiel für einen Zufallszahlengenerator, dessen Ausgabe laufend in einen Ringpuffer geschrieben wird und von dort bei Bedarf angefordert werden kann. Die grauen Quadrate symbolisieren die Information, dass die Zufallszahl bereits einmal gelesen wurde und kein weiteres Mal angefordert werden kann. Dieser Art von Puffer wird bei manchen langsam arbeitenden Zufallszahlengeneratoren eingesetzt.

Java-Beschleuniger (*Java accelerator*)

Java Card hat sich innerhalb von zwei Jahren zum Industriestandard für ausführbare Programme auf Chipkarten etabliert. Da die Java-VM jedoch den Bytecode interpretieren muss und nicht direkt ausführen kann, folgt daraus zwangsläufig eine Einbuße an Geschwindigkeit gegenüber nativen Maschinenbefehlen, die direkt vom Prozessor verarbeitet werden. Durch die große Verbreitung von Java auf Chipkarten wurde es aber für die Halbleiterhersteller attraktiv, Abhilfen zum Problem der Ausführungsgeschwindigkeit zu schaffen. Dabei werden momentan zwei Lösungsansätze verfolgt:

Im ersten Ansatz werden große Teile der Java-VM als dedizierte Hardware-Komponente zusätzlich zum eigentlichen Prozessor auf dem Chipkarten-Mikrocontroller untergebracht. Die Technik geht dabei in Richtung Pico-Java, d. h. zu einem realen Chip, der direkt den Java Bytecode abarbeiten kann. Nachteilig bei dieser Lösung ist, dass der Java-Prozessor zusätzliche Fläche neben dem eigentlichen Prozessor benötigt und eine vollständige Implementierung einer Java-VM verhältnismäßig aufwendig ist. Der Vorteil dieser Lösung liegt in der hohen Ausführungsgeschwindigkeit.

Bei der zweiten Lösungsvariante wird der eigentliche Befehlsatz des Prozessors um typische Java-Maschinenbefehle erweitert. Damit lassen sich dann von der Software-VM gesteuert unmittelbar Bytecodes vom erweiterten Prozessor abarbeiten. Realisiert wird diese Variante durch eine Lookup-Tabelle des Prozessors, welche für jeden zu emulierenden Bytecode die entsprechende Sequenz an Mikrobefehlen der CPU enthält. Der Vorteil gegenüber dem ersten Ansatz ist der geringere Flächenbedarf auf dem Chip bei allerdings etwas niedrigerer Ausführungsgeschwindigkeit.

Koprozessor für symmetrische Kryptoalgorithmen

Sowohl im Zahlungsverkehr als auch in der Telekommunikation wird bislang der DES als der Standard-Kryptoalgorithmus eingesetzt. Dieses große Marktpotenzial führt dazu, dass es sich für Halbleiterhersteller lohnt, Chipkarten-Mikrocontroller mit einer eigenen DES-Recheneinheit (*DES calculation unit*) auszurüsten. Dies ist im Prinzip nicht weiter schwierig, da der DES ursprünglich vor allem für eine Realisation in Hardware geschaffen wurde. Die größten Probleme bei der Vermarktung von DES-Recheneinheiten auf dem Mikrocontroller sind aber überdies nicht technischer Natur, sondern ausfuhrrechtlicher, da in vie-

len Ländern Bausteine mit schneller, hardwarebasierter DES-Verschlüsselung unter diverse Exportreglementierungen fallen.

Betrachtet man aber die Leistungsdaten von DES-Recheneinheiten für Chipkarten-Mikrocontroller, so sieht man mehr als deutlich die Vorzüge. Es werden dabei Größenordnungen von 75 µs für eine einfache DES-Operation und 150 µs für eine Triple-DES-Operation mit 2 Schlüsseln bei 3,5 MHz erreicht. Eine Taktfrequenzerhöhung bewirkt eine lineare Reduzierung der Rechenzeit. Zudem benötigt eine DES-Recheneinheit auf dem Mikrocontroller nur unwesentlich mehr Chipfläche als der ROM-Programmcode einer DES-Implementation, sodass dadurch auch kein zusätzlicher Flächenbedarf auf dem Chip entsteht.

In Zukunft wird es neben dem DES auch spezielle Koprozessoren für den AES auf Chipkarten-Mikroprozessoren geben. Dabei werden in der Regel alle drei möglichen Schlüssellängen 128 Byte, 196 Byte und 256 Byte unterstützt. Dies ist technisch genauso realisierbar wie DES-Koprozessoren, da der AES ebenfalls verhältnismäßig einfach in eine Hardwareschaltung umgesetzt werden kann.

Koprozessor für asymmetrische Kryptoalgorithmen

Für Berechnungen im Rahmen von Public-Key-Algorithmen, wie RSA und elliptischen Kurven, gibt es speziell entwickelte Recheneinheiten, die zusätzlich zu den üblichen Funktionseinheiten eines Chipkarten-Mikrocontrollers auf dem Silizium aufgebracht sind. Diese Recheneinheiten beherrschen nur einige Grundrechenarten, die für diese Art von Algorithmen notwendig sind. Dies ist zum einen die Exponentation und die Moduloberechnung großer Zahlen. Die Schnelligkeit dieser ganz auf diese beiden Rechenoperationen hin optimierten Bauteile kommt durch die sehr breite Architektur von bis zu 140 Bit zustande. In ihrem speziellen Einsatzgebiet sind sie zum Teil selbst einem leistungsfähigen PC überlegen.

Aufgerufen wird die Recheneinheit vom Prozessor, der entweder direkt die abzuarbeitenden Daten oder Zeiger auf die Daten übergibt und dann mit einem Befehl die Abarbeitung startet. Nachdem der Auftrag fertig gestellt und das Ergebnis im RAM abgelegt ist, erhält der Prozessor die Kontrolle über den Chip wieder zurück. Im Allgemeinen können die Koprozessoren beim RSA-Algorithmus alle Schlüssellängen bis 1 024 Bit und mittelfristig auch bis 2 048 Bit verarbeiten. Bei elliptischen Kurven sind derzeit bis zu 160 Bit und in naher Zukunft 210 Bit üblich.

Fehlererkennung und -korrektur im EEPROM

Die wesentliche Beschränkung der Lebensdauer von Chipkarten ist bestimmt durch das EEPROM und seine technisch begrenzte Anzahl der möglichen Schreib-/Löschzyklen. Um diese Einschränkung abzumindern, können beispielsweise Fehlerkorrekturcodes für bestimmte, stark beanspruchte EEPROM-Bereiche per Software errechnet und damit dann Fehler korrigiert werden. Es ist auch möglich, die Fehlerkorrekturcodes als Hardwareschaltung auf dem Chip zu implementieren. Damit werden, transparent für die Software, EEPROM-Fehler erkannt und, falls diese nicht zu umfangreich sind, korrigiert.

Zur Speicherung der Codes benötigt man natürlich ein zusätzliches EEPROM. Da gute Fehlerkorrekturcodes aber relativ viel Speicherplatz benötigen, steht man damit vor einer strategischen Entscheidung. Ist die Fehlerkorrektur gut, fordert dies einen Mehrverbrauch

von bis zu 50 % des zu schützenden Speichers. Der Speicher für die Fehlerkorrekturmechanismen kann jedoch nur für diesen Zweck verwendet werden. Benutzt man eine nicht so gute Fehlerkorrektur, ist der zusätzliche Aufwand an Speicher zwar geringer, doch der Nutzen sehr fraglich.

Es gibt wenige Mikrocontroller auf dem Markt, die eine in Hardware realisierte EEPROM-Fehlererkennung und -korrektur haben, die aber dazu unter Umständen die Hälfte des zu schützenden Speichers zusätzlich benötigen. Eine Konsequenz daraus ist die Tatsache, dass das dem Benutzer zur Verfügung stehende EEPROM nicht allzu groß ist. Allerdings ist die Lebensdauer des mit diesem Mechanismus abgesicherten EEPROMs ein Mehrfaches des üblichen Wertes.

Erweiterung der Chiphardware

Ist aus bestimmten Gründen eine Erweiterung der Chiphardware notwendig, so bedeutet dies einen großen Aufwand an Entwicklungszeit und Kosten beim Chiphersteller. Es gibt zur Realisierung einer kundenindividuellen Hardware nur zwei Möglichkeiten: Aufbau der neuen Hardware auf dem Silizium einer bestehenden Chip-Generation oder Konstruktion einer Zwei-Chip-Lösung mit allen ihren Nachteilen.

Um für dieses Problem einen akzeptablen Weg zu finden, hat man eine Kompromisslösung ersonnen. Diese wählt vom Prinzip her den Mittelweg der beiden Lösungsmöglichkeiten. Der Chip für die neue Hardware-Baugruppe wird dabei direkt mit dem bisherigen Chip verklebt und durch Bond-Drähte elektrisch verbunden. Dieser Lösung kommt zugute, dass die meisten Mikrocontroller für Chipkarten nicht nur eine I/O-Leitung haben, sondern mehrere, welche zur Kommunikation mit dem zusätzlichen Chip verwendet werden können. Die resultierende Dicke in Sandwichbauweise unterscheidet sich nur unwesentlich von normalen Chips, da man das Siliziumträgermaterial einfach stärker abschleift und es damit dünner wird. Damit kann man einen Sandwichchip ohne Zusatzaufwand in Standardmodule einbauen.

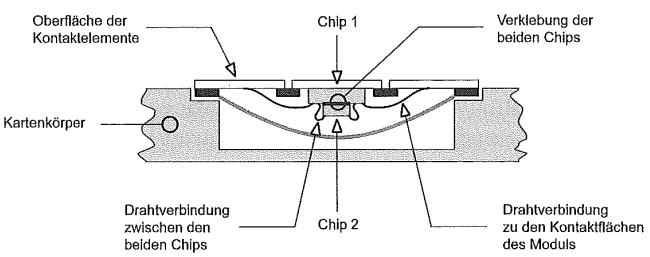


Bild 3.63 Querschnitt durch ein Chipmodul, das zwei verschiedene Chips enthält, die elektrisch durch Bonddrähte miteinander verbunden sind.

Das beschriebene Verfahren ist ideal dafür geeignet, kundenindividuelle Wünsche für eine zusätzliche Hardware ohne aufwendige Änderungen durchzuführen. Es lassen sich damit bestehende Chips mit neuen Baugruppen verbinden, die beispielsweise über eine besondere serielle Schnittstelle zur Prüfung von Sicherheitsmerkmalen an anderen Chips verfügen. Auch besteht dadurch die Möglichkeit, einen speziellen ASIC mit einem geheimen kryptografischen Algorithmus in eine Chipkarte einzubringen. Für große Stückzahlen im Millionenbereich ist das Verfahren nicht rentabel, da sich hierbei auch eine Entwicklung von

Spezialchips lohnt. Für kleinere bis mittlere Stückzahlen stellen aber Chips in Sandwichbauweise eine sehr gute Lösung für Vorserien oder Spezialanwendungen – wie Sicherheitsmodule für Terminals oder auch Chipkarten für Pay-TV-Decoder[6] – dar.

VSI (*vertical system integration*), face-to-face
Eine andere Form der Erweiterung der Chiphardware bzw. der auf einem einzelnen Chip unverträglicher Kombination von Halbleitertechnologien ist VSI (*vertical system integration*). Dabei sind zwei oder mehrere dünn geschliffene Dies in einem Stapel miteinander mechanisch und mittels halbleitertechnischer Durchkontaktierungen der einzelnen Dies auch elektrisch miteinander verbunden. Mit VSI kann auf technisch sehr elegante Weise die ursprüngliche Chipfläche in ganzen Vielfachen vergrößert werden. Bei zwei gestapelten Dies hat man bereits die doppelte Fläche zur Verfügung und beispielsweise bei vier Dies die vierfache Fläche. Dies ermöglich zum einen sowohl eine erhebliche Erweiterung des zur Verfügung stehenden Speichers und zum anderen auch eine beträchtliche Vergrößerung der Chipsicherheit. Es ist nämlich zurzeit *de facto* unmöglich, mit halbleitertechnischem Analysegerät einen zwischen zwei Dies eingestapelten Chip ohne Zerstörung des abdeckenden Chips anzugreifen.

Eine einfachere Variante der im Prinzip beinahe beliebig skalierbaren VSI ist die Face-to-face-Anordnung zweier Chips. Dabei wird die elektrische Kontaktierung durch das hochgenaue Aufeinandersetzen von zwei Chips mit ihrer Oberseite (*face*) geschaffen.

Sowohl VSI als auch das Face-to-face-Verbinden zweier Chips ermöglichen deutlich bessere Erweiterungen der Chiphardware, als dies durch das Verbinden zweier Chips durch Bonddrähte möglich ist.

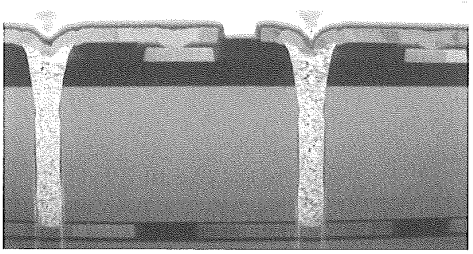


Bild 3.64 Querschnittfoto eines VSI-Stapels. Gut zu erkennen sind die beiden Durchkontaktierungen zwischen zwei aufeinandergestapelten Dies. (Giesecke & Devrient)

3.5 Kontaktbehaftete Karten

Der Hauptunterschied einer Chipkarte zu allen anderen Karten ist der implantierte Mikrocontroller. Findet seine Energieversorgung und die Datenübertragung kontaktbehaftet statt, so ist dazu eine galvanische Verbindung notwendig. Diese besteht aus den sechs oder acht vergoldeten Kontakten, die auf jeder üblichen Chipkarte zu sehen sind. Die Lage dieser Kontakte in Bezug zum Kartenkörper und ihre Größe ist in der ISO 7816-2 aus dem Jahr 1988 festgelegt.

[6] Siehe auch [Kuhn].

In Frankreich hatte man schon lange vor der ISO 7816-2 eine nationale Norm von AFNOR, die eine etwas höhere Position der Kontaktflächen als ISO vorsah. Diese ist auch in der ISO noch als „transitional contacts location" vorhanden, doch lautet die Empfehlung der Norm, diese Position in Zukunft nicht mehr zu verwenden. Allerdings gibt es in Frankreich noch viele Karten mit dieser Kontaktposition, sodass wohl mit einem Aussterben nicht so bald zu rechnen ist.

Die absolute Position der Kontaktflächen befindet sich in der linken oberen Ecke des Kartenkörpers. Die Maßzeichnung in Bild 3.65 verdeutlicht dies.

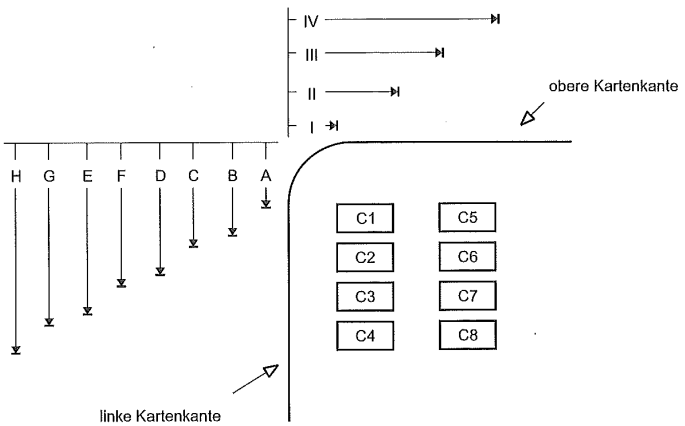


Bild 3.65 Position der Kontakte in Relation zum Kartenkörper. Die Position der Kontakte zum Kartenkörper ist nicht maßstabsgetreu.

I	maximal	10,25 mm	A	maximal	19,23 mm
II	minimal	12,25 mm	B	minimal	20,93 mm
III	maximal	17,87 mm	C	maximal	21,77 mm
IV	minimal	19,87 mm	D	minimal	23,47 mm
			E	maximal	24,31 mm
			F	minimal	26,01 mm
			G	maximal	26,85 mm
			H	minimal	28,55 mm

Die minimale Größe der Kontakte beträgt 1,7 mm in der Höhe und 2 mm in der Breite. Die maximale Größe der Kontakte ist nicht festgelegt. Sie ist allerdings durch die Vorgabe, die einzelnen Kontakte elektrisch voneinander zu isolieren, begrenzt.

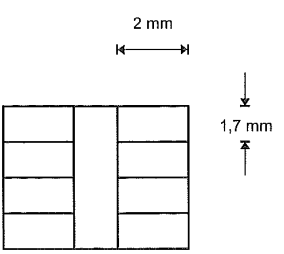


Bild 3.66 Die minimale Größe der Kontakte nach ISO 7816-2.

Die Position des Moduls auf dem Kartenkörper ist durch die Normen fest vorgegeben. Das Kartenelement Magnetstreifen und das Feld für die Hochprägung sind ebenfalls genau festgelegt (ISO 7811). Alle drei Elemente können sich auf einer Karte gleichzeitig befinden. Dabei müssen aber folgende Beziehungen zueinander beachtet werden: Befindet sich nur ein Chip und ein Prägefeld auf der Karte, so können sich diese sowohl auf der gleichen als auch auf verschiedenen Kartenseiten befinden. Ist zusätzlich noch ein Magnetstreifen vorhanden, muss sich dieser immer auf der gegenüberliegenden Seite des Prägefeldes befinden.

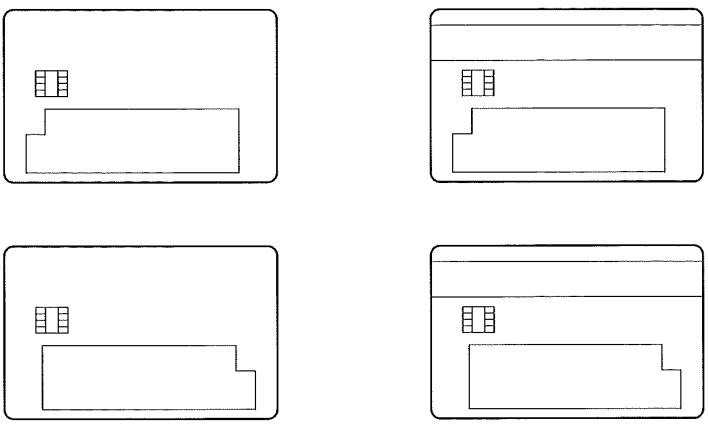


Bild 3.67 Die nach ISO 7816-2 erlaubten Varianten der Anordnung der Kartenelemente: Chip, Prägefeld und Magnetstreifen.

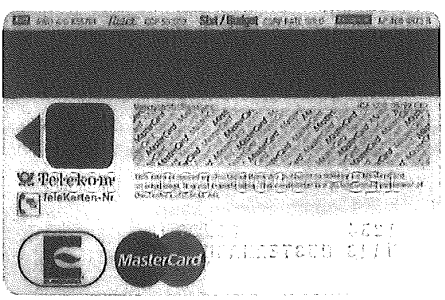


Bild 3.68 Beispiel für eine Karte mit Chip, Magnetstreifen, Unterschriftsfeld und Hochprägung. (Giesecke & Devrient)

3.6 Kontaktlose Karten

Wie bereits in Abschnitt 2.3.3 beschrieben, ist bei der kontaktlosen Karte keine galvanische Kopplung zwischen der Chipkarte und dem Kartenterminal erforderlich, um Energie und Daten über kurze Entfernungen zu übertragen. Die technischen Vorteile der kontaktlosen Kartentechnologie sowie die zusätzlichen Anwendungen, die hierdurch erschlossen

werden, sind ebenfalls in Abschnitt 2.3.3 dargestellt. In diesem Kapitel befassen wir uns ausführlicher mit der Technik und Funktionsweise der kontaktfreien Chipkarten. Die technischen Verfahren zur Energie- und Datenübertragung, die in den kontaktlosen Karten zum Einsatz kommen, sind nicht neu, sondern schon seit vielen Jahren von den so genannten RFID-Systemen (*Radio Frequency Identification*) allgemein bekannt und vielfältig genutzt, wie z. B. in Tierimplantaten oder in den Transpondern für die elektronische Wegfahrsperre in Kraftfahrzeugen. In der Funk- und insbesondere der Radartechnik kennt man vielfältige Möglichkeiten zur Identifizierung von Personen oder Objekten über kleine oder auch große Entfernungen. Diese Vielzahl der technischen Möglichkeiten ist für die Anwendung in Chipkarten im ID1-Format – und auf dieses wollen wir uns hier beschränken – stark reduziert, da alle Funktionselemente in dem nur 0,76 mm dicken und elastischen Kartenkörper untergebracht werden müssen. Ein ungelöstes technisches Problem ist z. B. nach wie vor der Einbau von flexiblen Batterien in den Kartenkörper in der Großserie. Es gibt zwar mittlerweile flexible Batterien in geeigneter Dicke. Erfahrungen aus der Massenproduktion sowie aus dem Feld liegen aber noch nicht vor. Deshalb ist man zurzeit nach wie vor auf die passiven Verfahren beschränkt, bei denen die Energie zur Versorgung der Karte aus dem elektromagnetischen Feld des Kartenterminals gewonnen werden muss, wodurch die Reichweite auf den Meterbereich eingeschränkt wird.

Um den Überblick über die Vielzahl der Verfahren zu erleichtern, kann man sie nach verschiedenen Klassifizierungsmerkmalen einteilen. Eine Möglichkeit ist die Einteilung nach der Art der Energie- und Datenübertragung. Die am häufigsten verwendeten Verfahren sind die Übertragung mit Radio- oder Mikrowellen, die optische Übertragung, die kapazitive und die induktive Kopplung. Für die flache Bauform der Chipkarte ohne eigene Stromversorgung eignet sich am ehesten die kapazitive und die induktive Kopplung. Die zurzeit im Markt angebotenen Systeme benutzen auch ausschließlich diese Verfahren, die auch als Einzige in den relevanten ISO/IEC-Normreihen 10 536, 14 443 und 15 693 berücksichtigt sind. Wir beschränken uns deshalb in diesem Buch ebenfalls auf die Darstellung dieser Verfahren.

Ebenso wie bei den kontaktbehafteten Chipkarten besteht ein System mit kontaktlosen Chipkarten mindestens aus zwei Komponenten, der Karte selber und einem dazu passenden Kartenterminal, welches je nach der eingesetzten Technologie als Lese- oder Schreiblesegerät fungiert. In der Regel beinhaltet das Terminal noch eine weitere Schnittstelle, über die es mit einem Hintergrundsystem kommunizieren kann.

Für die Kommunikation der kontaktlosen Karte mit dem Terminal müssen folgende vier Aufgaben gelöst werden:

- die Energieübertragung für die Versorgung der integrierten Schaltung
- die Übertragung des Taktsignals
- die Datenübertragung zur Chipkarte
- die Datenübertragung von der Chipkarte

Es sind viele verschiedene Konzepte für die Lösung dieser Probleme entwickelt worden, die auf die Erfahrungen der RFID-Technologie aufbauen und zumeist auf spezielle Anwendungen zugeschnitten sind. So ist es z. B. ein großer Unterschied, ob der Abstand zwischen dem Kartenterminal und der Karte beim Betrieb wenige Millimeter oder etwa einen

Meter beträgt. Die Entwicklung vieler unterschiedlicher Lösungen, die für spezielle Anwendungen zugeschnitten und optimiert sind, führt allerdings zwangsläufig zu deren Inkompatibilität.

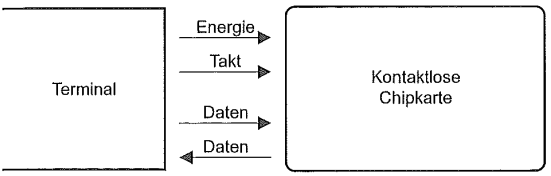


Bild 3.69 Die notwendige Energie- und Datenübertragung zwischen Terminal und kontaktloser Chipkarte.

Induktive Kopplung

Die induktive Kopplung ist zurzeit das am weitesten verbreitete Verfahren für kontaktlose Chipkarten, mit dem sich sowohl die Daten- wie auch die Energieübertragung realisieren lassen. Unterschiedliche Anforderungen und Randbedingungen, z. B. Funkzulassungsvorschriften, haben zu unterschiedlichen Realisierungen geführt.

Bei manchen Anwendungen wie z. B. der Zugangskontrolle kann es ausreichen, wenn die Daten in der Karte nur gelesen werden können, was technisch einfache Lösungen ermöglicht. Wegen der niedrigen Leistungsaufnahme von wenigen 10 µW geht die Reichweite dieser Karten bis in den Meterbereich. Die Speicherkapazität beträgt meistens nur einige 100 Bit. Sollen die Daten auch geschrieben werden, erhöht sich die Leistungsaufnahme auf mehr als 100 µW. Das hat zur Folge, dass die Reichweite im Schreibbetrieb auf ca. 10 cm begrenzt ist, da sich die abgestrahlte Leistung des Schreibgerätes wegen der Funkzulassungsvorschriften nicht beliebig erhöhen lässt. Kontaktlose Mikroprozessorkarten haben eine noch höhere Leistungsaufnahme von typisch 10 mW. Die Entfernung zum Terminal ist hierdurch noch enger begrenzt.

Unabhängig von Reichweite und Leistungsaufnahme arbeiten alle Karten mit induktiver Kopplung nach dem gleichen Prinzip.

Im Kartenkörper sind ein oder mehrere Chips sowie eine oder mehrere meist großflächige Spulen als Koppelelemente für die Energie- und Datenübertragung integriert.

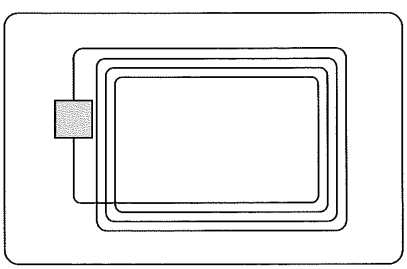


Bild 3.70 Prinzipieller Aufbau einer kontaktlosen Chipkarte mit induktiver Kopplung.

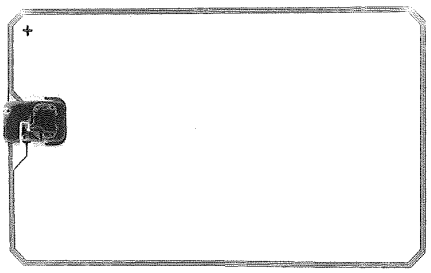


Bild 3.71 Inlay-Folie einer kontaktlosen Chipkarte mit induktiver Kopplung und geätzter Spule.

Energieübertragung

Kontaktlose Chipkarten werden fast ausnahmslos passiv betrieben. Das bedeutet, dass die gesamte Energie, die für den Betrieb des Chips in der Karte notwendig ist, vom Lesegerät in die Karte übertragen werden muss.

Die Energieübertragung geschieht nach dem Prinzip des lose gekoppelten Transformators. Von einer Spule des Terminals wird hierfür ein starkes elektromagnetisches Hochfrequenzfeld erzeugt. Die am häufigsten verwendeten Frequenzen sind dabei <135 kHz und 13,56 MHz, was Wellenlängen von 2400 m bzw. 22 m entspricht. Die Wellenlängen des elektromagnetischen Feldes sind somit um ein Vielfaches größer als der Abstand der Karte zum Terminal, das heißt, die Karte befindet sich im Nahfeld des Terminals, womit das Modell des lose gekoppelten Transformators zulässig ist. Bringt man eine kontaktlose Karte in die Nähe des Terminals, so durchdringt ein Teil des Magnetfeldes die Spule der Karte, sodass an ihr eine Wechselspannung U_i induziert wird. Diese Spannung wird gleichgerichtet und steht zur Energieversorgung des Chips zur Verfügung. Da die Kopplung zwischen den Spulen im Terminal und in der Karte nur sehr schwach ist, ist der Wirkungsgrad der beschriebenen Anordnung nur sehr klein. Zur Erzeugung der erforderlichen Feldstärken müssen deshalb in der Terminalspule sehr hohe Ströme fließen, was man dadurch erreicht, dass man parallel zur Spule L_T einen Kondensator C_T schaltet, dessen Kapazität so gewählt ist, dass er zusammen mit der Spuleninduktivität einen Parallelschwingkreis bildet, dessen Resonanzfrequenz der Übertragungsfrequenz zur Karte entspricht.

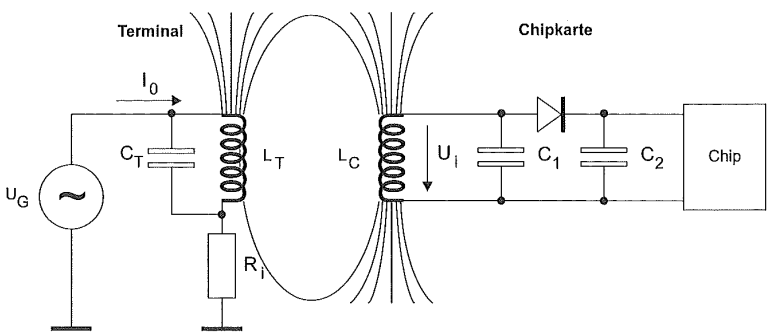


Bild 3.72 Spannungsversorgung einer Chipkarte mit induktiver Kopplung.

Die Spule L_C in der Karte bildet zusammen mit dem Kondensator C_1 ebenfalls einen Schwingkreis mit derselben Resonanzfrequenz. Die in der Karte induzierte Spannung ist proportional zur Übertragungsfrequenz, zur Windungszahl der Spule L_C sowie zur von der Spule umschlossenen Fläche. Das bedeutet, dass mit zunehmender Übertragungsfrequenz die Anzahl der erforderlichen Windungen in der Karte abnimmt (bei 125 kHz 100 bis 1000 Windungen, bei 13,56 MHz 3 bis 10 Windungen).

Datenübertragung

Für die Datenübertragung vom Terminal zur Karte können alle bekannten digitalen Modulationsverfahren eingesetzt werden. Die gebräuchlichsten sind:

- ASK *(Amplitude Shift Keying)*
- FSK *(Frequency Shift Keying)*
- PSK *(Phase Shift Keying)*

Wegen der besonders einfachen Demodulationsmöglichkeiten werden meist die ASK- oder die PSK-Modulation benutzt.

In der umgekehrten Richtung, von der Chipkarte zum Terminal, benutzt man eine Art Amplitudenmodulation, die durch eine vom Datensignal gesteuerte Änderung der Last in der Karte erzeugt wird (Lastmodulation). Bringt man eine Chipkarte, die auf die Resonanzfrequenz des Terminals abgestimmt ist, in das Nahfeld des Terminals, so entzieht diese wie oben beschrieben dem magnetischen Feld Energie. Hierdurch erhöht sich des Strom I_0 in der Koppelspule des Terminals, was als erhöhter Spannungsabfall am Innenwiderstand R_i detektiert werden kann. Durch Ändern der Last an der Spule der Chipkarte, z. B. durch Zu- und Abschalten des Lastwiderstandes R_2, kann somit die Spannung U_0 im Terminal durch die Chipkarte amplitudenmoduliert werden. Wenn man das Zu- und Abschalten des Widerstandes R_2 durch Daten steuert, so können diese Daten im Terminal detektiert und ausgewertet werden.

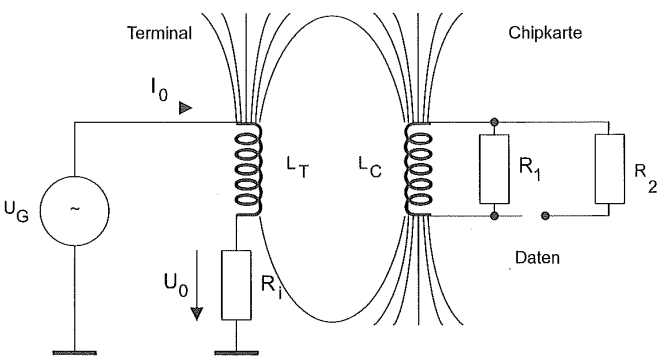


Bild 3.73 Beispiel für das Prinzip der Lastmodulation zur Datenübertragung bei der kontaktlosen Chipkarte.

Wegen der geringen Kopplung der Spulen im Terminal und in der Karte sind die durch diese Lastmodulation erzeugten Spannungsänderungen im Terminal sehr klein. In der Praxis ergibt sich ein Nutzsignal von wenigen mV, welches nur durch aufwendige Schaltungen detektiert werden kann, da es dem gleichfrequenten wesentlich größeren (ca. 80 dB)

Sendesignal des Terminals überlagert ist. Verwendet man jedoch einen Hilfsträger mit der Frequenz f_H, so erscheint das empfangene Datensignal im Terminal auf den beiden Seitenbändern mit den Frequenzen $f_{Leser} \pm f_H$, das durch Filterung mit einem Bandpass vom wesentlich stärkeren Signal des Terminals getrennt und verstärkt werden kann. Die Demodulation ist dann problemlos möglich.

Die Modulation mit Hilfsträger hat aber den Nachteil, dass sie eine wesentlich größere Bandbreite benötigt als die direkte Modulation. Daher kann dieses Verfahren nur in wenigen Frequenzbereichen eingesetzt werden.

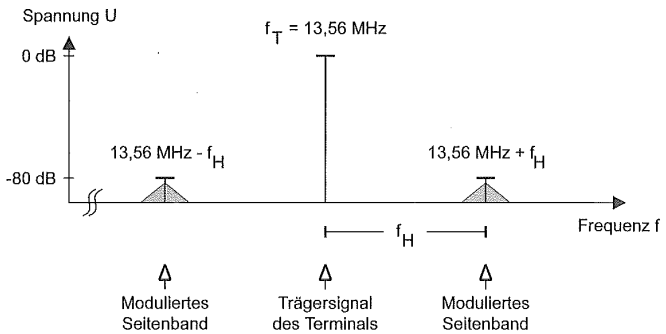


Bild 3.74 Durch eine Lastmodulation mit Hilfsträger entstehen zwei Seitenbänder im Abstand der Hilfsträgerfrequenz f_H um die Sendefrequenz des Lesegerätes. Die eigentliche Information steckt in den Seitenbändern der beiden Hilfsträger-Seitenbänder, welche durch die Modulation des Hilfsträgers selbst entstehen (nach Klaus Finkenzeller [Finkenzeller 02]).

Kapazitive Kopplung

Bei geringem Abstand zwischen Karte und Kartenterminal besteht die Möglichkeit, die Daten durch kapazitive Kopplung zu übertragen. Bei der kapazitiven Kopplung sind leitende Flächen im Kartenkörper und im Kartenterminal so angebracht, dass sie beim Einführen

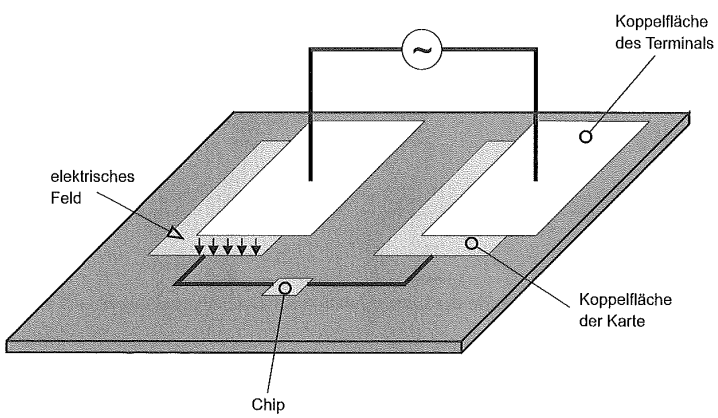


Bild 3.75 Prinzip der kapazitiven Kopplung. Die Kopplung entsteht durch das elektrische Wechselfeld zwischen den parallelen elektrisch leitenden Flächen in der Karte und dem Terminal.

oder Auflegen der Karte als Kondensatorplatten wirken. Die erreichbare Kapazität hängt dabei im Wesentlichen von der Größe der Koppelflächen und von deren Abstand ab. Die maximale Fläche ist dabei durch die Abmessungen der Karte begrenzt. Der minimale Abstand wird durch die notwendige Isolation zwischen den Koppelflächen bestimmt. Mit vertretbarem Aufwand ist eine nutzbare Koppelkapazität von einigen 10 pF erreichbar. Dies reicht für die Energieübertragung eines Mikroprozessors nicht aus. Deshalb wird diese Methode nur zur Datenübertragung verwendet, während die Energieübertragung induktiv erfolgt. Dieses gemischte Verfahren ist in der ISO/IEC 10 536 für „close coupling cards" standardisiert und im folgenden Kapitel ausführlich beschrieben. Wie der Name schon sagt, ist das Verfahren auf kleine Koppelabstände begrenzt.

Antikollision

Beim Betrieb von kontaktlosen Karten kann man nicht ausschließen, dass sich mehrere Karten gleichzeitig in der Reichweite des Terminals befinden. Das gilt insbesondere für Systeme mit größerer Reichweite. Aber selbst bei Systemen mit geringer Reichweite kann es vorkommen, dass z. B. zwei Karten direkt übereinander liegen und vom Terminal gleichzeitig aktiviert werden. Alle Karten, die sich in der Reichweite eines Terminals befinden, werden gleichzeitig versuchen, auf Kommandos des Terminals zu reagieren. Die gleichzeitige Datenübertragung von mehreren Karten führt aber zwangsläufig zu Störungen und Datenverlusten, wenn nicht gezielt Gegenmaßnahmen ergriffen werden. Die technischen Verfahren, welche einen störungsfreien Datenaustausch mit mehreren Karten im Aktionsradius eines Kartenterminals sicherstellen, nennt man Antikollisionsverfahren.

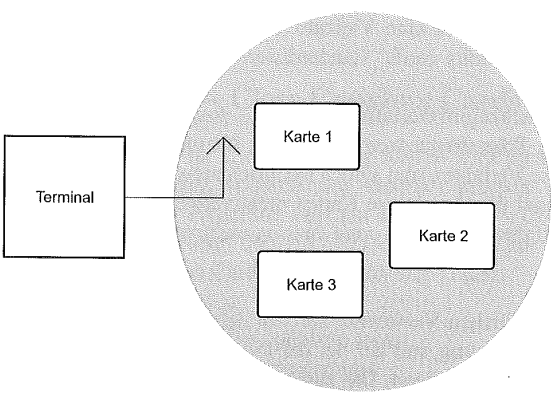


Bild 3.76 Der gleichzeitige Betrieb von mehreren Karten in der Reichweite eines Terminals (Vielfachzugriff) erfordert ein Antikollisionsverfahren zum störungsfreien Datenaustausch.

In der Kommunikationstechnik ist der Datenaustausch zwischen vielen Teilnehmern und einer Basisstation ein häufig auftretendes Problem und wird als Vielfachzugriff bezeichnet. Ein typisches Beispiel sind die Mobiltelefonnetze, bei denen alle Teilnehmer, die sich in einer Funkzelle befinden, gleichzeitig auf eine Basisstation zugreifen. Es wurden zahlreiche Verfahren entwickelt, die es ermöglichen, die Signale der einzelnen Teilnehmer zu trennen. Diese Antikollisionsverfahren lassen sich in vier Gruppen aufteilen.

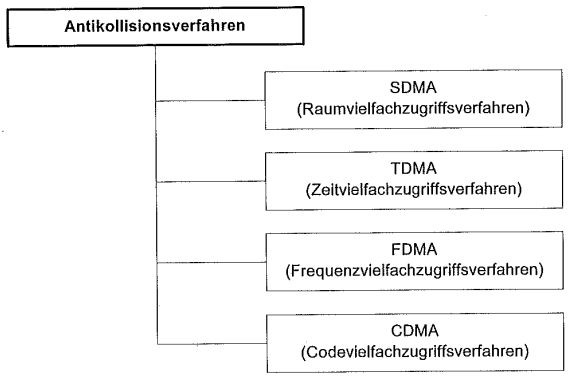


Bild 3.77 Einteilung der Antikollisionsverfahren nach den vier Grundprinzipien.

Beim Raumvielfachzugriffsverfahren (SDMA – *space division multiple access*) wird versucht, den Operationsbereich des Terminals so einzugrenzen oder auch abzutasten, dass immer nur eine Karte gleichzeitig erfasst werden kann. Da diese Technik einen sehr hohen Aufwand bei der Implementierung der Antennen erfordert, kommt sie für kontaktlose Karten nicht zur Anwendung.

Beim Zeitvielfachzugriffsverfahren (TDMA – *time division multiple access*) wird dafür gesorgt, dass die Karten ein unterschiedliches Zeitverhalten haben und dadurch vom Terminal identifiziert und einzeln angesprochen werden können. Diese Verfahren werden am häufigsten eingesetzt, und es gibt eine Vielzahl von Varianten. Zwei davon sind in der ISO/IEC 14 443-3 für „proximity cards" standardisiert und werden in Abschnitt 3.6.3 ausführlich beschrieben.

Bei den Frequenzvielfachzugriffsverfahren (FDMA – *frequency division multiple access*) werden unterschiedliche Trägerfrequenzen gleichzeitig für mehrere Übertragungskanäle zur Verfügung gestellt. Diese Verfahren sind jedoch technisch aufwendig und deswegen teuer. Sie werden aus diesem Grund für kontaktlose Karten nicht verwendet. Dieses gilt auch für das Codevielfachzugriffsverfahren (CDMA – *code division multiple access*).

Stand der Normung

Bei den vielen unterschiedlichen Verfahren, die von den einzelnen Herstellern verwendet wurden, war die Standardisierung, mit der bei der ISO/IEC 1988 begonnen wurde, erwartungsgemäß schwierig und zeitraubend. Die zuständige Arbeitsgruppe hat die Aufgabe, eine kontaktlose Karte zu normen, die weitgehend kompatibel zu den anderen Standards für Identifikationskarten ist. Das heißt, dass eine kontaktlose Karte auch andere Funktionselemente wie Magnetstreifen, Hochprägung und Chipkontakte haben kann. Hierdurch wird der gleichzeitige Einsatz der kontaktlosen Technik in bestehenden Systemen mit anderer Technik ermöglicht. Wie bereits erläutert, hängen die technischen Möglichkeiten der kontaktlosen Energie- und Datenübertragung wesentlich von dem gewünschten Abstand zwischen Karte und Terminal beim Lesen und Schreiben ab. Es war deshalb auch nicht möglich, eine einzige Norm zu schaffen, die alle aus den unterschiedlichen Anwendungen resultierenden Anforderungen mit einer einzigen technischen Lösung erfüllt.

Zurzeit sind drei verschiedenen Normen fertig gestellt, die unterschiedliche Lesereichweiten beschreiben. In jeder dieser Normen sind wiederum unterschiedliche technische Lösungen zugelassen, da man sich in dem Normungsgremium nicht auf ein einziges Verfahren einigen konnte. Um die Interoperabilität zwischen den unterschiedlichen Varianten herzustellen, müssen die Kartenterminals alle Varianten unterstützen.

Tabelle 3.7 Fertiggestellte ISO/IEC-Normen für kontaktlose Chipkarten.

Norm	Typ der kontaktlosen Chipkarten	Reichweite
ISO/IEC 10536	close coupling	bis ca. 1 cm
ISO/IEC 14443	proximity coupling, PICC	bis ca. 10 cm
ISO/IEC 15693	vicinity coupling, VICC	bis ca. 1 m

Begonnen wurde zunächst mit der Standardisierung der „Close Coupling Cards" (ISO/IEC 10536), weil die zum damaligen Zeitpunkt verfügbaren Mikroprozessoren einen relativ hohen Stromverbrauch hatten, sodass eine Energieübertragung über größere Abstände noch nicht möglich war. Die wesentlichen Teile dieser Norm sind fertig gestellt und verabschiedet und werden im folgenden Abschnitt beschrieben. Dieser Kartentyp bietet in der Handhabung nur geringe Vorteile gegenüber den kontaktbehafteten Karten, da die Karte zum Betrieb in ein Kartenterminal gesteckt oder zumindest auf einer Auflage des Kartenterminals exakt positioniert werden muss. Darüber hinaus ist der Kartenaufbau komplex, was zu hohen Herstellkosten führt. Deshalb konnten sich Systeme dieses Typs bisher kaum im Markt durchsetzen.

3.6.1 ISO/IEC 10536 – Close Coupling Cards

In der Norm ISO/IEC 10536 für Close-Coupling-Karten wird diese Anwendung als „slot or surface operation" bezeichnet, wodurch zum Ausdruck gebracht wird, dass die Karte im Betrieb in einen Kartenschlitz eingeführt oder auf eine definierte Oberfläche des Terminals aufgelegt werden muss. Die Norm ISO/IEC 10536 mit dem Titel „Identification Cards – Contactless Integrated Circuit(s) Cards" besteht aus vier Teilen:

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

Die Teile 1 bis 3 sind inzwischen internationale Norm, Teil 4 ist noch in Bearbeitung. Wichtige Rahmenbedingungen für diese Norm waren:

- weitgehende Kompatibilität zu ISO 7816
- Betrieb bei beliebiger Orientierung der Karte zum Leser
- Trägerfrequenz der Übertragung zwischen 3 und 5 MHz
- Bidirektionale Datenübertragung mit induktiver oder kapazitiver Kopplung
- Leistungsaufnahme der Karte kleiner als 150 mW (ausreichend für den Betrieb von Mikroprozessoren)

In Teil 1 der oben genannten Norm sind die physikalischen Eigenschaften definiert. Es werden im Wesentlichen die gleichen Anforderungen festgelegt wie für Chipkarten mit Kontakten, insbesondere auch hinsichtlich Biegung und Torsion. Ein Unterschied besteht in der Belastbarkeit durch elektrostatische Entladungen. Da bei einer kontaktlosen Karte keine leitenden Verbindungen zwischen der Kartenoberfläche und den integrierten Schaltungen im Kartenkörper erforderlich sind, ist eine kontaktlose Karte weitgehend unempfindlich gegen ESD-Schäden. Entsprechend ist in der Norm eine Testspannung von 10 kV festgelegt, im Vergleich zu 1,5 kV bei kontaktbehafteten Karten.

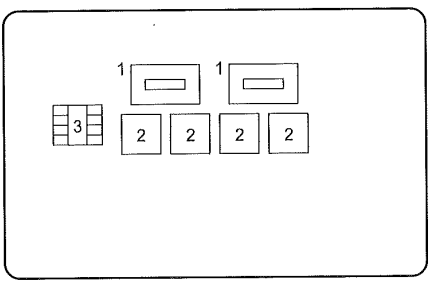


Bild 3.78 Anordnung der Koppelelemente in kontaktlosen Chipkarten: Koppelspulen in der Chipkarte (1), Kapazitive Koppelflächen in der Chipkarte (2) und Kontaktflächen des Chips (3).

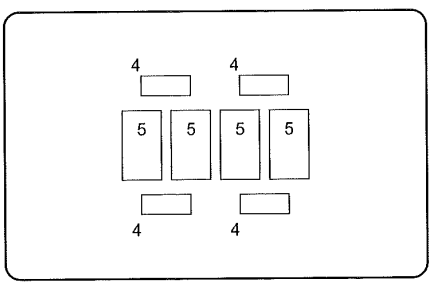


Bild 3.79 Anordnung der Koppelelemente bei Terminals für kontaktlose Chipkarten: Koppelspulen im Terminal (4) und kapazitive Koppelflächen im Terminal (5).

Im Teil 2 von ISO/IEC 10536 sind die Lage und die Abmessungen der Koppelelemente spezifiziert. Da man sich nicht auf ein einziges Verfahren einigen konnte, sind sowohl kapazitive wie auch induktive Koppelelemente in der Art definiert, dass beide Techniken gleichzeitig in der Karte und im Terminal implementiert werden können. Ein Beispiel hierfür ist in den Bildern 3.78 und 3.79 gezeigt.

Die Anordnung ist so gewählt worden, dass die Lageunabhängigkeit bei entsprechender Ansteuerung im Terminal gegeben ist.

Der Teil 3 von ISO/IEC 10536 wurde 1996 veröffentlicht. Er ist der bisher wichtigste Teil der Norm und beschreibt die Modulationsverfahren für eine induktive wie auch kapazitive Datenübertragung, da man sich nicht auf ein einziges Verfahren einigen konnte. Ein

normgerechtes Terminal muss daher beide Verfahren unterstützen. Es können auch beide Verfahren in einer Karte implementiert werden.

Energieübertragung

Die Energieübertragung erfolgt durch ein sinusförmiges magnetisches Wechselfeld mit einer Frequenz von 4,9152 MHz, welches eine oder mehrere der induktiven Koppelflächen durchsetzt, je nachdem, wie viele Koppelspulen in der Karte vorhanden sind. Das Terminal muss alle vier Felder erzeugen.

Die magnetischen Wechselfelder F1 und F2 durch die Flächen H1 und H2 haben dabei eine Phasendifferenz von 180°, ebenso wie die Felder F3 und F4 durch die Flächen H3 und H4. Zwischen den Feldern F1 und F3 und ebenso zwischen F2 und F4 besteht eine Phasendifferenz von 90°. Jedes einzelne Magnetfeld ist so stark, dass es mindestens 150 mW in die Karte übertragen kann. Insgesamt soll die Karte aber nicht mehr als 200 mW Leistung aufnehmen. Diese komplizierte Definition der Magnetfelder ist erforderlich, um die induktive Datenübertragung bei 4 Lagen Invarianz zu ermöglichen, wie weiter unten erläutert wird.

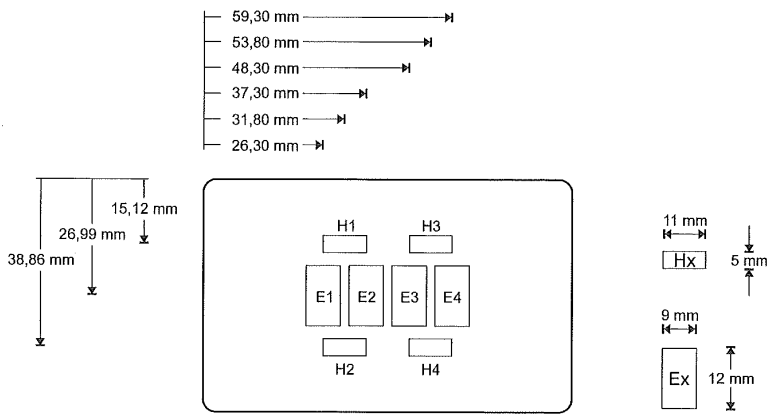


Bild 3.80 Die Lage und Maße der Koppelflächen in der kontaktlosen Karte und im Terminal.

3.6.1.1 Induktive Datenübertragung

Für die induktive Datenübertragung werden unterschiedliche Modulationsverfahren für die beiden Übertragungsrichtungen angewendet.

Übertragung von der Karte zum Terminal

Für die Übertragung der Daten von der Karte zum Terminal wird zunächst ein Hilfsträger durch Lastmodulation (siehe Bild 3.81) mit einer Frequenz von 307,2 kHz erzeugt, wobei die Last um mindestens 10 % geändert wird. Die Datenmodulation erfolgt dann durch Phasenumschaltung des Hilfsträgers von 180°, wodurch sich zwei Zustände der Phase ergeben, die als logisch 0 und 1 interpretiert werden können. Der Anfangszustand nach Aufbau des magnetischen Feldes wird als logisch 1 definiert, wobei dieser Anfangszustand (Zeitintervall t_3 in Bild 3.84) mindestens 2 ms stabil bleibt. Danach bedeutet jeder Phasensprung des Hilfsträgers eine Umkehrung des logischen Zustandes, sodass sich eine NRZ-Codierung (*non return to zero*) ergibt. Die Übertragungsrate beträgt zumindest für den ATR 9 600 Bit/s.

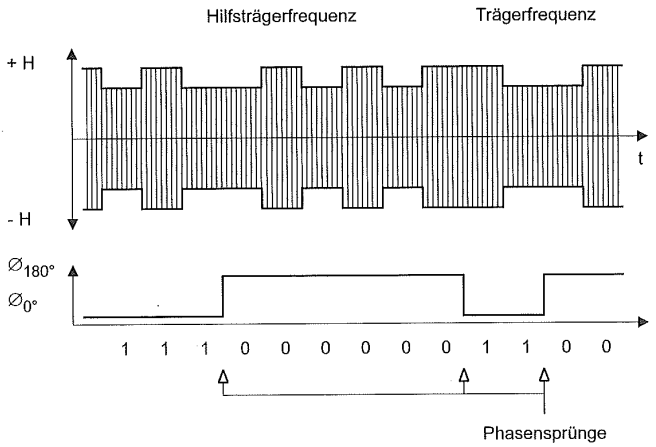


Bild 3.81 Der zeitliche Ablauf beim Prinzip der Phasenmodulation bei der Datenübertragung mit einer kontaktlosen Chipkarte. Die obere Grafik zeigt das magnetische Wechselfeld und die untere Grafik die dazugehörige Phasenlage. Die Trägerfrequenz ist 4,9152 MHz und die Hilfsträgerfrequenz 307,2 kHz.

Übertragung vom Terminal zur Karte

Zur Datenübertragung vom Terminal zur Karte werden die vier magnetischen Wechselfelder F1 bis F4, welche die vier Koppelflächen H1 bis H4 durchdringen, phasenmoduliert (PSK – *Phase Shift Keying*), und zwar so, dass die Phase aller Felder gleichzeitig um 90° springt. Hierdurch werden zwei Phasenzustände A und A′ definiert. Abhängig von der Orientierung der Karte zum Terminal ergeben sich aus Sicht der Karte zwei unterschiedliche Konstellationen für die Phasenzustände, die in den Bildern 3.82 und 3.83 dargestellt sind.

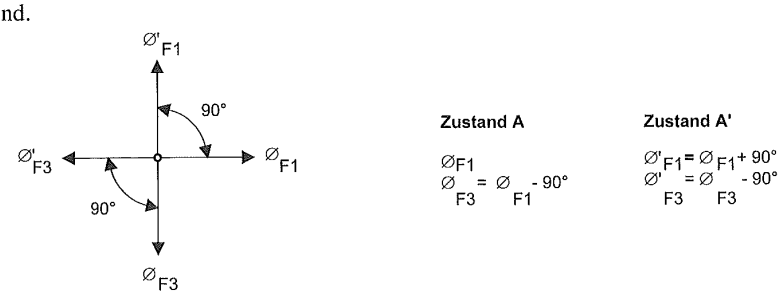


Bild 3.82 Erste Variante der Phasenmodulation bei der Datenübertragung mit einer kontaktlosen Chipkarte. Die eingezeichneten Pfeile sind als Phasenvektoren aufzufassen.

Da die Karte in allen vier möglichen Lagen in Relation zum Terminal funktionieren soll, wird unabhängig von der gegebenen Alternative der Anfangszustand (im Zeitintervall t_2 und t_3 in Bild 3.83) als logisch 1 interpretiert. Danach bedeutet jeder Phasensprung eine Umkehrung des logischen Zustandes, sodass sich wiederum eine NRZ-Codierung ergibt.

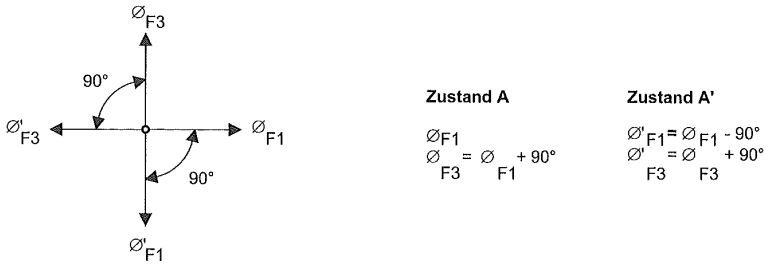


Bild 3.83 Zweite Variante der Phasenmodulation bei der Datenübertragung mit einer kontaktlosen Chipkarte. Die eingezeichneten Pfeile sind als Phasenvektoren aufzufassen.

3.6.1.2 Kapazitive Datenübertragung

Für die kapazitive Datenübertragung von der Karte zum Terminal wird ein Paar der Koppelflächen entweder E1 und E2 oder E3 und E4 benutzt (Bild 3.80), abhängig davon, in welcher Lage zum Terminal sich die Karte befindet. Das andere Flächenpaar kann für die Datenübertragung in umgekehrter Richtung genutzt werden. Da die Karte den ATR über eines der beiden Flächenpaare sendet, kann das Terminal die Orientierung der Karte erkennen. Die maximale Spannungsdifferenz zwischen einem Flächenpaar ist auf 10 V begrenzt, sie muss aber mindestens so groß sein, dass die minimale Differenzspannungsschwelle des Empfängers von 300 mV überschritten wird. Für die Datenübertragung wird eine differenzielle NRZ-Codierung verwendet. Der Sender schaltet hierzu die Spannung zwischen den Flächen E1 und E2 oder zwischen E3 und E4 um. Der Zustand für logisch 1 ist wiederum im Zeitintervall t_3 (Bild 3.84) festgelegt. Danach bedeutet jede Umpolung der Spannung einen Wechsel des logischen Zustandes.

Anfangszustand und Answer to Reset

Damit das Terminal am Anfang eines Datenaustausches die Art der Datenübertragung und die Lage der Karte eindeutig feststellen kann, müssen Zeitintervalle für den Beginn der Energie- und Datenübertragung definiert sein. In Bild 3.84 sind die Bedingungen und Werte für die Reset-Erholzeit t_0, die Leistungsanstiegszeit t_1, die Vorbereitungszeit t_2, die Zeit für den stabilen logischen Zustand t_3 und die Zeit für den Answer to Reset t_4 dargestellt.

Minimum der Reset-Erholzeit – t_0

Wenn ein Reset durch Ab- und Wiederanschalten der energieübertragenden Felder erfolgen soll, muss die Zeit zwischen Ab- und Anschalten der Felder, während der keine Energie übertragen wird, größer oder gleich 8 ms sein.

Maximum der Leistungsanstiegszeit – t_1

Die Anstiegszeit des energieübertragenden Feldes, welches durch das Terminal aufgebaut wird, muss kleiner oder gleich 0,2 ms sein.

Vorbereitungszeit – t_2

Die Vorbereitungszeit für die Karte, um in einen stabilen Zustand zu kommen, beträgt 8 ms.

Zeit für den stabilen logischen Zustand – t_3

Vor dem Answer to Reset wird der logische Zustand für eine Zeit von 2 ms auf 1 gehalten. Während dieser Zeit werden Karte und Terminal für die induktive Datenübertragung auf logisch 1 gesetzt.

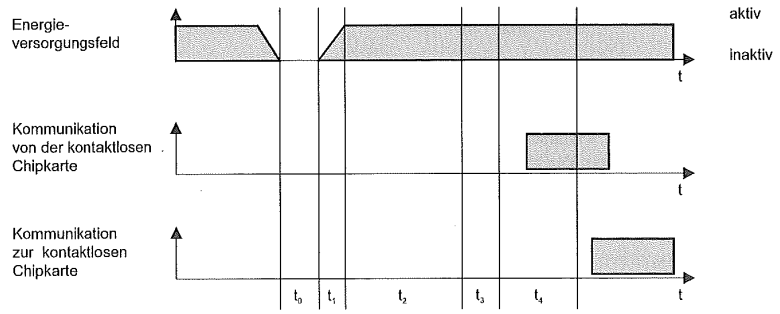


Bild 3.84 Zeitlicher Ablauf der Datenübertragung bei einer kontaktlosen Chipkarte nach ISO/IEC 10 536-3 mit $t_0 \geq 8$ ms, $t_1 \leq 0{,}2$ ms, $t_2 = 8$ ms, $t_3 = 2$ ms, $t_4 \leq 30$ ms.

Maximale Antwortzeit für ATR – t_4

Die Karte muss mit dem Senden des ATR vor Ablauf von 30 ms beginnen. Die Karte kann im ATR anzeigen, dass die Bedingungen für den folgenden Betrieb hinsichtlich Energiepegel, Datenübertragungsrate oder Frequenz der Felder geändert werden müssen. Der hierdurch gegebene Freiraum kann den Erfordernissen der Anwendung entsprechend genutzt werden. So können z. B. für zeitkritische Anwendungen wesentlich höhere Übertragungsraten gewählt werden.

3.6.2 Remote-Coupling-Karten

Unter dem Begriff „Karten mit Remote Coupling" werden die Chipkarten zusammengefasst, die über eine Entfernung von einigen Zentimetern bis zu etwa 1 Meter zum Terminal Daten übertragen können. Diese Möglichkeit ist bei allen Anwendungen von großem Interesse, bei denen Daten zwischen Karte und Terminal ausgetauscht werden sollen, ohne dass der Karteninhaber die Karte in die Hand nehmen und in ein Terminal einführen muss. Beispielhafte Anwendungen sind:

- Zugangskontrolle
- Fahrzeugerkennung
- elektronischer Fahrschein
- Skipass
- Flugticket
- elektronische Geldbörse
- Gepäckerkennung

Die Vielfalt der Anwendungen lässt erahnen, dass es eine große Vielfalt an möglichen technischen Realisierungen gibt. Bei der Normung wurde versucht, die Vielfalt der techni-

schen Varianten zu begrenzen, was nur zum Teil gelungen ist. Die internationalen Normen ISO/IEC 14 443 und ISO/IEC 15 633 decken die Entfernungsbereiche um 10 cm sowie um einen Meter ab.

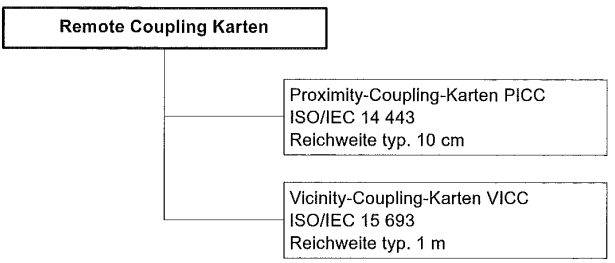


Bild 3.85 Die ISO/IEC Normen unterscheiden bei den Remote-Coupling-Karten zwischen den Proximity-Karten und den Vicinity-Karten.

3.6.3 Proximity Integrated Circuit(s) Cards nach ISO/IEC 14 443

Die ISO/IEC 14443 hat den Titel „Identification cards – Contactless integrated circuit(s) cards – Proximity cards" und beschreibt die Eigenschaften und die Funktionsweise von kontaktlosen Chipkarten mit einer Reichweite von ca. 10 cm. Die auf diese Entfernung mögliche Energieübertragung reicht zur Versorgung eines Mikroprozessors aus. Um diese Karten auch in der vorhandenen Infrastruktur für kontaktbehaftete Chipkarten nutzen zu können, besitzen die Karten häufig neben den Koppelelementen zum kontaktfreien Betrieb zusätzlich Kontakte, sodass wahlweise ein Betrieb über Kontakte oder kontaktlos möglich ist. Diese Karten werden Dual-Interface-Karten oder auch Combikarte genannt.
Die ISO/IEC 14443 besteht aus folgenden Teilen:

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

Physikalische Eigenschaften
Die physikalischen Eigenschaften, die im Teil 1 der ISO/IEC für Proximity-Karten (PICC – *Proximity Integrated Circuit(s) Cards*) definiert sind, entsprechen im Wesentlichen den Anforderungen, wie sie auch für Chipkarten mit Kontakten festgelegt sind. Beim Gebrauch dieser Karten ist damit zu rechnen, dass sie elektromagnetischen Feldern ausgesetzt werden, die für den Betrieb anderer Kartentypen, die z. B. den Normen ISO/IEC 10536 oder ISO/IEC 15 693 entsprechen, vorgesehen sind. Dies und auch der Einfluss üblicher Umweltbelastung durch elektromagnetische Felder darf nicht zu dauerhaften Schäden an den Karten führen. Um dies sicherzustellen, sind in der Norm Maximalwerte für die Belastung durch magnetische und elektrische Wechselfelder festgelegt, welche die Karte ohne Schaden aushalten muss. Es ist Aufgabe der Halbleiterhersteller, das Design der Chips so zu gestalten, dass diese Anforderungen erfüllt werden.

Radio frequency power and signal interface

Die Proximity-Karten arbeiten nach dem Prinzip der induktiven Kopplung. Sowohl die Energieversorgung als auch der Datenaustausch erfolgen über das magnetische Wechselfeld, das vom Kartenterminal erzeugt wird. In der ISO/IEC 14 443 wird das Kartenterminal mit „proximity coupling device" (PCD) bezeichnet. Der besseren Lesbarkeit wegen wird im Folgenden weiter der allgemeine Begriff Kartenterminal neben der Abkürzung PCD verwendet.

Die Sendefrequenz des PCD ist auf $f_C = 13{,}56$ MHz mit einer Toleranz von plus oder minus 7 kHz festgelegt mit einer Magnetfeldstärke H von minimal 1,5 A/m und maximal 7,5 A/m (Effektivwerte). Der typische Verlauf der Feldstärke H eines PCD in Abhängigkeit von der Entfernung x ist in Bild 3.86 dargestellt.

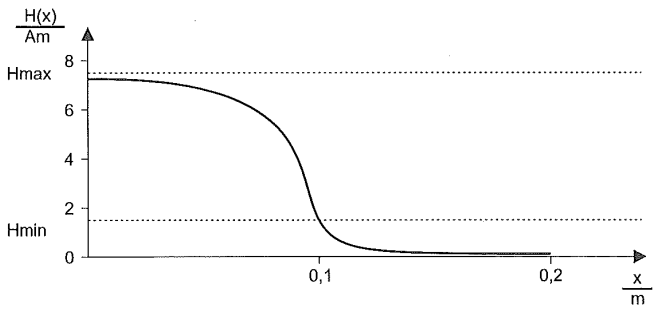


Bild 3.86 Typischer Feldstärkeverlauf eines Kartenterminals für Proximity-Karten (PCD)

Aus dem Feldstärkeverlauf des PCD sowie der Ansprechfeldstärke einer Proximity-Karte (PICC) kann man die Reichweite des Systems abschätzen. Mit dem typischen Feldstärkeverlauf aus Bild 3.86 und einer angenommenen Ansprechfeldstärke der PICC von 1,5 A/m ergibt sich z. B. eine Reichweite von ca. 10 cm.

Signal- und Kommunikationsinterface

In der ISO/IEC 14 443 sind zwei verschiedene Kommunikationsschnittstellen definiert, die mit Typ A und Typ B bezeichnet werden. Der Grund für die Normung von zwei unterschiedlichen Verfahren ist nicht technischer Art. Vielmehr gab es zu der Zeit, als die ISO/IEC 14 443 erarbeitet wurde, bereits unterschiedliche technische Konzepte von verschiedenen Herstellern. Wegen unterschiedlicher Interessenlagen der Beteiligten konnte man sich – wie leider häufiger in der Normung – nicht auf ein einziges Verfahren einigen, was technisch sinnvoll gewesen wäre. Man einigte sich als Kompromiss auf die oben genannten beiden Verfahren, die dann letztendlich im Juni 2001 als internationale Norm veröffentlicht wurden. Schon bei den bereits vorhandenen Verfahren muss ein Kartenterminal beide Verfahren unterstützen, um volle Interoperabilität mit allen Karten nach ISO/IEC 14443 zu erreichen, da die Karten in der Regel nur ein Verfahren unterstützen.

Während das Terminal darauf wartet, eine Proximity-Karte zu entdecken, muss es periodisch zwischen den beiden Kommunikationsverfahren umschalten. Hierdurch kann es sowohl Typ A wie auch Typ B Karten erkennen. Wenn das PCD eine Karte erkannt hat, bleibt es in dem entsprechenden Kommunikationsverfahren, bis die Karte vom Terminal deaktiviert oder die Karte aus dem Arbeitsbereich des Terminals entfernt wird.

3.6.3.1 Kommunikationsinterface Typ A

Bei den Typ-A-Karten geschieht die Datenübertragung in beiden Richtungen mit einer Bitrate von $f_C/128$ ($\approx$106 kBit/s).

Datenübertragung vom Terminal zur Karte

Für die Datenübertragung vom PCD zur Karte wird eine digitale Amplitudenmodulation (100 % ASK) mit modifizierter Millercodierung verwendet. Die Länge der Austastlücke (Pause) ist dabei auf maximal 3 µs begrenzt. Diese relativ kurze Austastzeit erleichtert eine stetige Energieversorgung der Karte. Die genaue Festlegung der Dauer und des Ein- und Ausschwingverhaltens des Pause-Signals ist in Bild 3.87 angegeben.

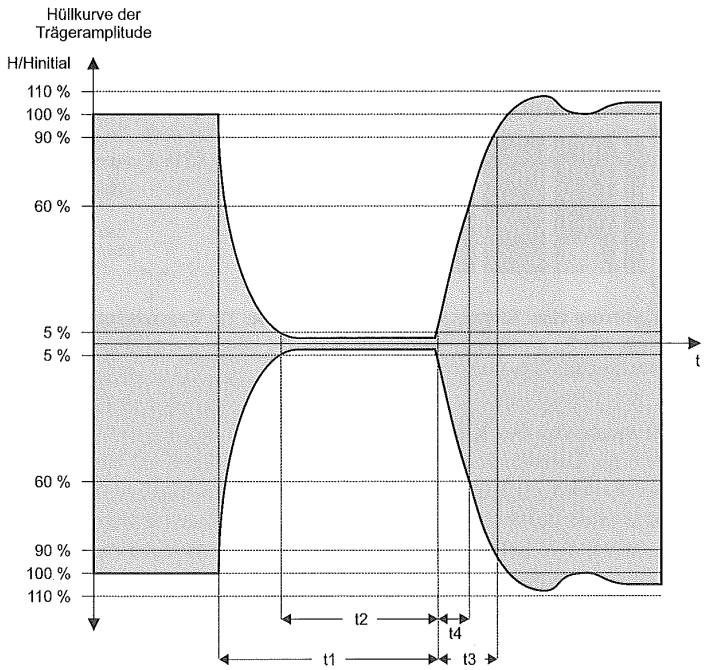


Bild 3.87 Spezifikation der Austastlücke (Pause) nach ISO/IEC 14443-2. Die maximale Länge der Pause ist auf 3 µs begrenzt, um die Energieübertragung zur Karte nur so kurz wie möglich zu unterbrechen. Es gilt: 2,0 µs $\leq$ t1 $\leq$ 3,0 µs, falls t1 > 2,5 µs dann gilt 0,5 µs $\leq$ t2 $\leq$ t1, falls t1 $\leq$ 2,5 µs dann gilt 0,7 µs $\leq$ t2 $\leq$ t1, 0 µs $\leq$ t3 $\leq$ 1,5 µs und 0 µs $\leq$ t4 $\leq$ 0,4 µs.

Die Karte erkennt das Ende der Pause während des Zeitintervalls t_4, das heißt, nachdem das magnetische Feld 5 % und bevor es 60 % von $H_{INITIAL}$ überschritten hat. Die Überschwinger müssen auf $H_{INITIAL} \pm 10$ % begrenzt sein.

Ein Beispiel für die Codierung einer Bitfolge nach dem Verfahren der modifizierten Millercodierung ist in Bild 3.88 dargestellt.

Dabei gelten folgende Codierungsregeln:

- logisch „1“: Austastlücke nach der halben Bitdauer.
- logisch „0“: keine Austastung mit folgenden beiden Ausnahmen:
  - Falls zwei oder mehr logisch „0“ aufeinander folgen, gibt es eine Austastlücke zu Beginn der Bitdauer.
  - Falls das erste Bit eines Protokollrahmens eine „0“ ist, wird diese durch eine Austastlücke zu Beginn der Bitdauer dargestellt.
- Beginn einer Kommunikation: Austastlücke am Beginn einer Bitdauer.
- Ende einer Kommunikation: Logisch „0“ gefolgt von einem Bit ohne Austastlücke.
- Keine Information: Mindestens für die Dauer von 2 Bit keine Austastlücke.

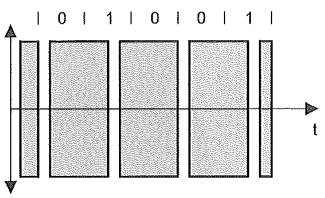


Bild 3.88 Codierung einer Bitfolge vom Terminal zur Karte für Kommunikationsinterface Typ A mit ASK 100 %, modifizierte Millercodierung, 106 kBit/s. Darstellung der Spannungen an der Antenne des Terminals.

Datenübertragung von der Karte zum Terminal

Die Bitrate für die Datenübertragung von der Karte zum Terminal beträgt ebenfalls $f_C/128$ (~106 kBit/s). Es wird ein Lastmodulationsverfahren mit Hilfsträger eingesetzt, das heißt, der Hilfsträger wird durch Schalten einer Last in der Karte erzeugt. Die Frequenz des Hilfsträgers ist auf $f_S = f_C/16$ (~847 kHz) festgelegt. Die Modulation des Hilfsträgers erfolgt durch Aus- und Einschalten des Hilfsträgers (On/Off Keying, OOK), wobei die Manchester-Codierung zum Einsatz kommt. Ein Beispiel für die Codierung einer Bitfolge ist in Bild 3.89 dargestellt.

Die Bitcodierung ist folgendermaßen definiert:

- logisch „1“: Der Träger wird während der ersten Hälfte der Bitdauer mit dem Hilfsträger moduliert.
- logisch „0“: Der Träger wird während der zweiten Hälfte der Bitdauer mit dem Hilfsträger moduliert.
- Beginn einer Kommunikation: Der Träger wird während der ersten Hälfte der Bitdauer mit dem Hilfsträger moduliert.
- Ende einer Kommunikation: Der Träger wird während einer Bitdauer nicht moduliert.
- Keine Information: Keine Modulation mit dem Hilfsträger.

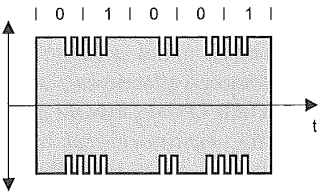


Bild 3.89 Lastmodulation für die Datenübertragung von der Karte zum Terminal mit einem Hilfsträger von $f_C/16$ (~847 kHz) und Manchester-Codierung mit einer Bitrate von 106 kBit/s und OOK. Darstellung der Spannungen an der Spule der Karte.

3.6.3.2 Kommunikationsinterface Typ B

Datenübertragung vom Terminal zur Karte

Bei den Typ-B-Karten wird für die Datenübertragung vom Terminal zur Karte eine ASK-Modulation mit einem Modulationsindex von 10 % (–2 %, +4 %) verwendet. Während bei dem Typ-A-Verfahren eine kontinuierliche Energieversorgung durch die sehr kurzen Austastlücken erreicht wird, ermöglicht dies beim Typ-B-Verfahren der kleine Modulationsindex, der so definiert ist, dass mindestens 86 % des Trägerfeldes zur Verfügung steht. Die Bitrate beträgt wiederum $f_C/128$ (~106 kBit/s). Die genaue Form der Typ-B-Modulation ist in Bild 3.90 dargestellt.

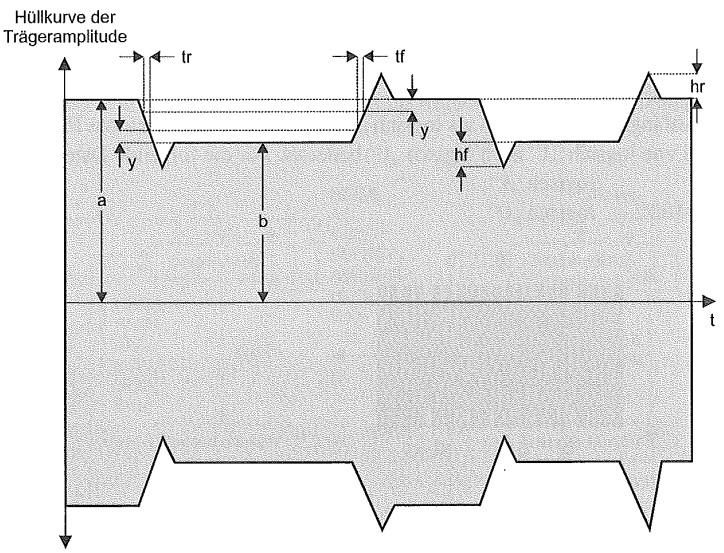


Bild 3.90 Modulation der Trägerfrequenz Typ B. Der geringe Modulationsindex von 10 % ermöglicht eine stetige Energieübertragung. Es gilt: tf, tr ≤ 2 µs, y = 0,1 • (a–b) und hf, hr ≤ 0,1 • (a–b).

Für die Bitcodierung wird die einfache NRZ (*Non-Return to Zero*) Codierung verwendet mit folgenden Codierungsregeln:

- logisch „1“: Große Amplitude der Trägerfrequenz.
- logisch „0“: Kleine Amplitude der Trägerfrequenz.

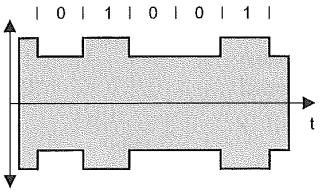


Bild 3.91 Codierung einer Bitfolge vom Terminal zur Karte für Kommunikationsinterface Typ B mit 10 % ASK, NRZ-Codierung und einer Bitrate von 106 kBit/s. Darstellung der Spannungen an der Antenne des Terminals.

Datenübertragung von der Karte zum Terminal

Für die Datenübertragung von der Karte zum Terminal wird auch bei Typ B die Lastmodulation mit Hilfsträger verwendet. Die Frequenz des Hilfsträgers f_s beträgt ebenso $f_C/16$ (~847 kHz). Die Modulation des Hilfsträgers erfolgt im Unterschied zu Typ A durch eine 180°-Phasenumtastung (*Binary Phase Shift Keying* – BPSK) des Hilfsträgers mit einer Bitrate von wiederum $f_C/128$ (~106 kBit/s) und der NRZ-Codierung. Um einen eindeutigen Anfangszustand zu haben, ist zu Beginn eines jeden Protokollrahmens folgende Sequenz einzuhalten:

- Für eine Zeit von TRO > $64/f_S$ *(guard time)* nach Empfang vom Terminal wird kein Hilfsträger erzeugt.
- Anschließend erzeugt die Karte für eine Zeitdauer von TR1 > $80/f_S$ (*synchronization time*) den Hilfsträger ohne Phasenumtastung. Die Phasenlage während dieser Zeit wird als Referenzphase ∅0 definiert.

Die Anfangsphase ∅0 wird als logisch „1" definiert, sodass der erste Phasensprung einen Wechsel von logisch „1" nach logisch „0" bedeutet. Für die folgende Datenübertragung gilt:

∅0: logisch „1"
∅0+180°: logisch „0".

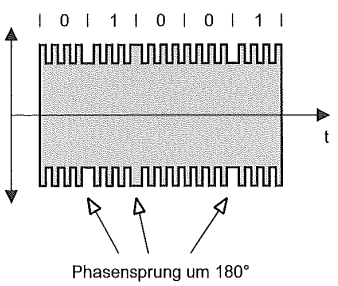


Bild 3.92 Codierung einer Bitfolge von der Karte zum Terminal für Kommunikationsinterface Typ B mit einem Hilfsträger von $f_C/16$ (~847 kHz), BPSK, NRZ Codierung und einer Bitrate von 106 kBit/s. Darstellung der Spannungen an der Spule der Karte.

3.6.3.3 Initialisierung und Antikollision (ISO/IEC 14 443-3)

Wenn eine Proximity-Karte in den Arbeitsbereich eines Terminals gelangt, muss zunächst eine Kommunikation zwischen Karte und Terminal aufgebaut werden. Hierbei kann es vorkommen, dass bereits eine Kommunikation mit einer anderen Karte besteht, oder dass

mehrere Karten gleichzeitig in den Aktionsradius des Terminals gelangen. Es müssen Mittel bereitgestellt werden, um in einem solchen Fall eine störungsfreie Kommunikation mit einer Karte bzw. eine gezielte Auswahl einer Karte zu ermöglichen.

Im Teil 3 von ISO/IEC 14 443 sind der Aufbau der Kommunikation zwischen Karte und Terminal sowie die verwendeten Antikollisionsverfahren zur Selektion einer einzelnen Karte beschrieben. Wegen der unterschiedlichen Modulationsverfahren unterscheiden sich auch die Protokollrahmen und die Antikollisionsverfahren von Typ-A- und Typ-B-Karten.

3.6.3.3.1 Initialisierung und Antikollision Typ A

Für die Initialisierung und Selektion von Typ-A-Karten wird ein dynamischer „Binary-Search-Algorithmus" verwendet. Bei diesem Verfahren ist es erforderlich, dass das Lesegerät eine Datenkollision auf Bitebene erkennen kann. Wie weiter unten erläutert wird, ermöglicht die verwendete Manchester-Codierung eine solche bitweise Kollisionserkennung (siehe auch Bild 3.97). Voraussetzung hierfür ist allerdings, dass alle Karten im Arbeitsbereich des Terminals ihre Daten synchron senden.

Gelangt eine Proximity-Karte in das Feld eines Terminals, wird der Mikroprozessor der Karte mit Energie versorgt und nach dem Power-On-Reset gelangt die Karte in den IDLE Status. In diesem Status darf die Karte nur auf ein REQA (Request Kommando Typ A) oder ein WUPA (Wake-UP Kommando Typ A) reagieren. Alle anderen Kommandos, die das Terminal aussendet, um mit eventuell bereits im Arbeitsbereich vorhandenen Karten von Typ A oder B zu kommunizieren, müssen ignoriert werden, damit diese Kommunikation nicht gestört wird. Das folgende Zustandsdiagramm zeigt alle möglichen Zustände einer Karte vom Typ A, die diese während der Initialisierungs- und Antikollisionsphase einnehmen kann.

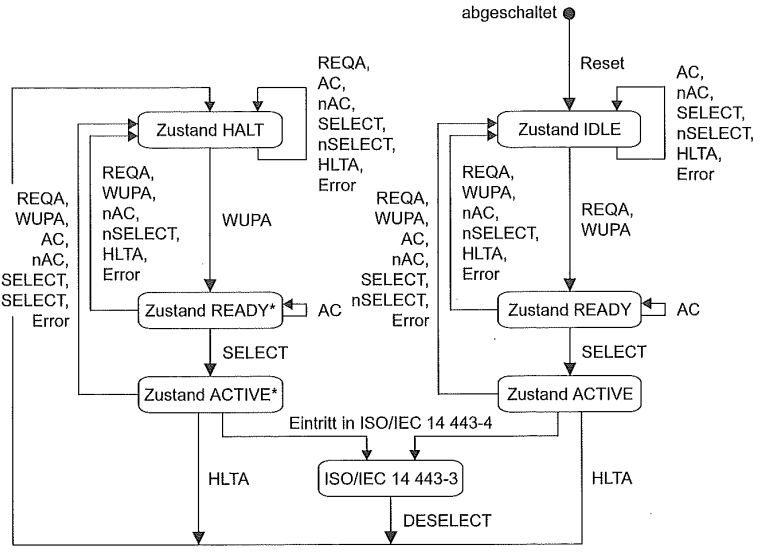


Bild 3.93 Zustandsdiagramm einer PICC Typ A während der Initialisierung und Antikollision. Die Erläuterung der Abkürzungen befindet sich im Text.

Wie bereits erwähnt, gelangt die Karte nach dem Power On Reset in den IDLE Status. Die Norm verlangt, dass die Karte den IDLE-Status innerhalb von 5 ms einnimmt, nachdem sie durch das Feld des Terminals ausreichend mit Energie versorgt wird. Im IDLE-Status horcht die Karte auf alle Kommandos und wechselt in den READY-Status, wenn sie ein REQA- oder WUPA-Kommando erkennt. Alle anderen Kommandos werden ignoriert.

Um eine hohe Zuverlässigkeit bei der Erkennung der Kommandos REQA und WUPA sicherzustellen, werden diese in speziellen kurzen Protokollrahmen (*short frames*) übertragen. Alle anderen Kommandos werden in Standardprotokollrahmen (*standard frames*) übertragen mit Ausnahme der Antikollisionskommandos, für die ein spezieller Bit-orientierter Rahmen (*Bit oriented anticollision frame*) definiert ist.

Short frame

Wie bereits erläutert, werden „short frames" nur für die Kommandos zur Initialisierung verwendet. Ein „short frame" besteht aus 9 Bit in folgender Reihenfolge:

- Start der Kommunikation,
- 7 Datenbits beginnend mit dem niederwertigsten Bit (*lsb first*),
- Ende der Kommunikation.

Die Codierungsregeln für das Start- und Ende Bit sowie für die Datenbits wurden bereits in Abschnitt 3.6.3.1 beschrieben.

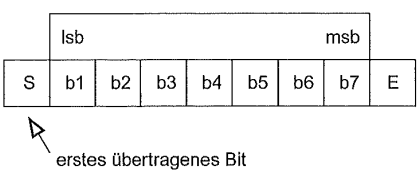


Bild 3.94 Aufbau eines „short frame".

Tabelle 3.8 zeigt die Codierung der REQA- und WUPA-Kommandos, die als Einzige in einem short frame gesendet werden.

Tabelle 3.8 Codierung der Kommandos REQA und WUPA, die das Short-Frame-Format mit 7 Datenbit benutzen.

b7	b6	b5	b4	b3	b2	b1	Bedeutung
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' = optionale Timeslot Methode
1	0	0	x	x	x	x	'40' . . . '4F' = proprietär
1	1	1	1	x	x	x	'78' . . . '7F' = proprietär
alle anderen Werte							RFU

Die Kommandos REQA und WUPA werden vom Terminal gesendet, um festzustellen ob sich Karten im Aktionsfeld des Terminals befinden (siehe auch Bild 3.93).

Standard frame

Standard frames werden für den regulären Datenaustausch verwendet und bestehen aus:

- Start der Kommunikation
- n • (8 Datenbits + ungerades Paritätsbit) mit n ≥ 1.
- Ende der Kommunikation

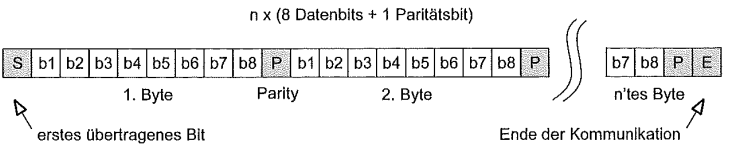


Bild 3.95 Aufbau eines „standard frame".

Wenn die Karte in den READY-Status wechselt, sendet sie nach der genau festgelegten frame delay time (siehe auch Bild 3.98) den „ Answer To Request, Typ A" (ATQA) aus. Der ATQA besteht aus zwei Byte und wird wegen der eindeutig festgelegten frame delay time synchron von allen angesprochenen Karten gesendet. Die Codierung von ATQA zeigt Bild 3.96.

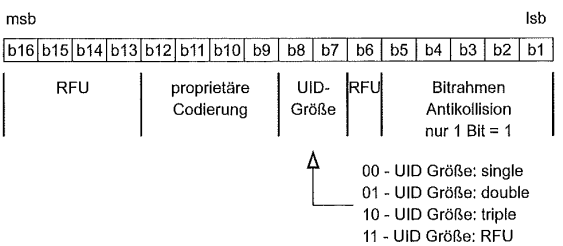


Bild 3.96 Codierung von ATQA. Alle RFU-Bit sind auf 0 zu setzen. Durch Bit 9 bis 12 können weiter nicht standardisierte Methoden angezeigt werden. Eines der Bit b1-b5 muss auf 1 gesetzt werden. Bit b7 und b8 geben die UID-Größe an.

Das Terminal erkennt am ATQA, dass sich mindestens eine Karte im Arbeitsbereich befindet, und startet das Antikollisionsverfahren und das Verfahren zum Lesen der Identifikationsnummer (UID, *Unique Identifier* Typ A) durch Senden eines SELECT-Kommandos. Hat das Terminal die vollständige Identifikationsnummer ermittelt, so sendet es ein SELECT-Kommando mit dieser Identifikationsnummer. Die Karte mit der entsprechenden Nummer bestätigt dieses Kommando mit einem SELECT-Acknowledge (SAK) und wechselt in den ACTIVE-Status. Im ACTIVE-Status kann die Karte nun mit Protokollen höherer Schichten (wie z. B. in ISO/IEC 14443-4 definiert) kommunizieren.

Durch ein HALT Kommando (HLTA, *Halt Command* Type A) wird die Karte in den HALT-Status versetzt. Die Karte kann auch mit speziellen Kommandos aus einer höheren Protokollschicht in den HALT-Status versetzt werden. Im HALT-Status reagiert die Karte nur auf ein Wake-up-Kommando (WUPA, *Wake-UP Command* Typ A), auf welches sie mit Senden des ATQA (*Answer To Request* Typ A) und einen Wechsel in den READY*-Status reagiert. Der READY*-Status ähnelt dem READY-Status. Die Bedingungen für den Wechsel in den ACTIVE*-Status können dem Bild 3.93 entnommen werden.

Das Antikollisionsverfahren sowie die Ermittlung der Identifikationsnummer sind im Folgenden ausführlich beschrieben.

Wenn sich zwei oder mehr Karten gleichzeitig im READY-Status im Arbeitsbereich des Terminals aufhalten, reagieren diese gleichzeitig auf ein SELECT-Kommando des Terminals durch Senden eines Teils ihrer unterschiedlichen Identifikationsnummern. Hierzu wird ein bitorientierter Frame verwendet, der es ermöglicht, dass nach einer beliebigen Anzahl gesendeter Datenbit die Übertragungsrichtung zwischen Terminal und Karte umgekehrt werden kann. Werden von mehreren Karten unterschiedliche Daten gesendet, empfängt das Terminal eine Überlagerung dieser Daten und erkennt eine Kollision daran, dass bei der Überlagerung von unterschiedlichen Bitmustern während der gesamten Bitdauer der Träger mit dem Hilfsträger moduliert wird. Dies ist ein irregulärer Zustand, da bei der verwendeten Manchester-Codierung immer eine Flanke während der Bitdauer vorkommen muss. In Bild 3.97 ist das Zustandekommen diese irregulären Zustandes dargestellt.

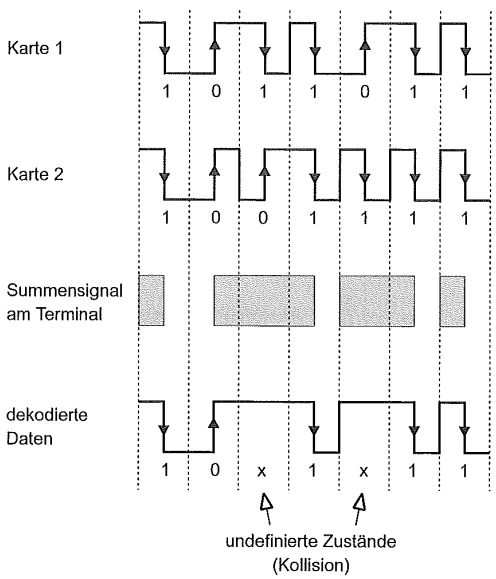


Bild 3.97 Kollision zweier Bitfolgen in Manchester-Codierung (Typ A). Bei ungestörter Übertragung wird der Träger immer nur während einer Hälfte der Bitdauer mit dem Hilfsträger moduliert. Bei der Überlagerung verschiedener Bits tritt eine Modulation während der gesamten Bitdauer auf, woran das Terminal eine Kollision erkennen kann.

Voraussetzung für das Erkennen einer Kollision auf Bitebene ist, dass alle im Aktionsfeld des Terminals befindlichen Karten, die sich im READY Status befinden, genau gleichzeitig auf ein ANTICOLLISION-Kommando antworten. Um dies sicherzustellen, ist das Zeitverhalten von Terminal und Karte beim Austausch von „frames" in der ISO/IEC 14 443-3 genau festgelegt.

Verzögerungszeit zwischen zwei Rahmen (*frame delay time – FDT*)
Die Zeit zwischen dem Ende der letzten Pause, die vom Terminal beim Ende der Kommunikation gesendet wurde, und der ersten Flanke der Modulation im Startbit der Karte wird

als „frame delay time PCD to PICC" mit der Abkürzung FDT bezeichnet und ist in 3.98 definiert. Es sind dabei zwei Fälle zu unterscheiden, je nachdem ob das letzte Datenbit, welches vom Terminal gesendet wurde, eine logische „1" oder eine logische „0" war.

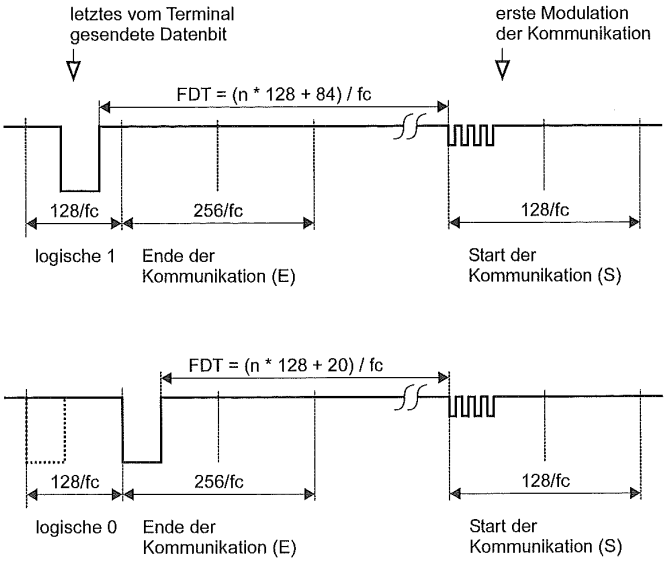


Bild 3.98 Rahmenverzögerungszeit zwischen Terminal und Karte (*frame delay time PCD to PICC* – FDT).

Für die Kommandos REQA, WUPA, ANTICOLLISION und SELECT ist der Wert von n auf 9 festgelegt, das heißt: die FDT ist $1\,236/f_C$ bzw. $1\,172/f_C$. Hierdurch wird erreicht, dass bei diesen Kommandos, die während der Antikollisionsschleife verwendet werden, alle Karten, die sich im Aktionsfeld befinden, synchron antworten. Hierdurch wird die Erkennung von Kollisionen auf Bitebene ermöglicht. Für alle anderen Kommandos gilt $n \geq 9$.

Die Zeit zwischen der letzten von der Karte gesendeten Modulation und dem ersten vom Terminal gesendeten Pause-Signal wird als „frame delay time PICC to PCD" bezeichnet. Diese Zeit ist für das Erkennen von Kollisionen auf Bitebene nicht von Bedeutung. Deshalb wird diese Zeit in der Norm auch nicht exakt definiert. Sie muss lediglich größer oder gleich $1172/f_C$ sein.

Außerdem ist noch die minimale Zeit zwischen zwei aufeinander folgenden REQA-Kommandos auf $7000/f_C$ festgelegt. Dies Zeit wird mit „Request guard time" bezeichnet.

Wie wir in den folgenden Abschnitten sehen werden, muss das Terminal für den Fall, dass sich mehrere PICCs im Aktionsfeld des Terminals befinden, unter Umständen viele Antikollisionsschleifen durchlaufen, bevor es eine einzelne Karte gezielt ansprechen kann. Um beim Durchlaufen dieser Schleifen nicht zu viel Zeit zu verbrauchen, wird für das Antikollisionskommando AC, welches eine spezielle Form des SELECT Kommandos ist, ein spezieller bitorientierter Protokollrahmen verwendet (*Bit oriented anticollision frame*). Dieser wird im Folgenden näher beschrieben.

Bit oriented anticollision frame

Entdeckt das Terminal (PCD) eine Kollision, reagiert es mit dem Senden von „bit oriented anticollision frames", die wie „standard frames" mit einer Länge von 7 Byte aufgebaut, aber in zwei Teile aufgeteilt sind: einen Teil 1 für die Übertragung vom PCD zum PICC und einen Teil 2 für die Gegenrichtung vom PICC zum PCD. Die Länge der einzelnen Teile verändert sich in den verschiedenen Antikollisionsschleifen, die nacheinander durchlaufen werden. Die Summe der Datenbits in beiden Teilen bleibt mit 56 immer konstant. Für die Länge von Teil 1 und Teil 2 gelten folgende Regeln:

- Regel 1: Die Summe der Datenbits ist 56.
- Regel 2: Die minimale Länge von Teil 1 beträgt 16 Datenbits.
- Regel 3: Die maximale Länge von Teil 1 beträgt 55 Datenbits.

Folglich beträgt die minimale Länge von Teil 2 ein Datenbit und die maximale Länge 40 Datenbits.

Die Trennung des Rahmens in zwei Teile kann an einer beliebigen Stelle erfolgen. Es kann also auch vorkommen, dass die Aufteilung mitten in einem Datenbyte erfolgt. In diesem Falle wird kein Paritätsbit an den ersten Teil des aufgeteilten Bytes angehängt. Das Paritätsbit des zweiten Teils des aufgeteilten Bytes wird vom PCD in diesem Falle ignoriert. In Bild 3.99 sind zwei Beispiele für einen Antikollisionsrahmen dargestellt.

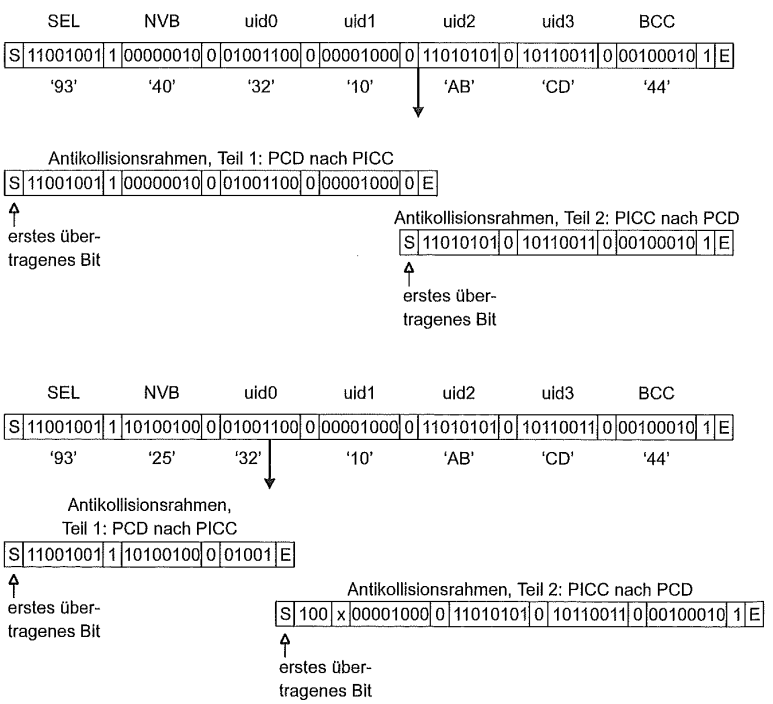


Bild 3.99 Zwei Beispiele für bitorientierte Antikollisionsrahmen. Im ersten Fall erfolgt die Trennung nach einem vollständigen Datenbyte, im zweiten Fall nach dem fünften Bit eines Datenbytes.

Kommandos während der Antikollisionsschleife
Während der Initialisierung und Antikollisionsschleife können, wie bereits aus dem Zustandsdiagramm in 3.93 hervorgeht, folgende Kommandos vorkommen:

- REQA (*Request command*, Typ A),
- WUPA (*Wake-up command*, Typ A),
- ANTICOLLISION,
- SELECT,
- HLTA (*Halt command*, Typ A).

Die Kommandos ANTICOLLISION und SELECT
Die Kommandos ANTICOLLSION und SELECT werden während der Antikollisionsschleife verwendet. Wie bereits aus 3.99 hervorgeht, bestehen diese Kommandos aus folgenden Datenfeldern:

- SEL, select code (1 Byte),
- NVB, Anzahl gültiger Bits (1 Byte),
- 0 bis 40 Datenbits der Identifikationsnummer (UID), abhängig vom Wert von NVB.

Die Identifikationsnummer (*unique identifier* – UID) kann aus 4 Byte (*single size*), 7 Byte (*double size*) oder 10 Byte (*triple size*) bestehen.

Tabelle 3.9 Mögliche Größen von UID (*Unique identifier*)

Kaskade Level	UID-Größe	Anzahl der UID-Bytes
1	Single	4
2	Double	7
3	Triple	10

Die UID kann eine feste Zahl oder eine Zufallszahl sein, die von der Karte erzeugt wird. Bei Double- oder Triple-size-Identifikationsnummern wird in Kaskade Level 1 und 2 als erstes Byte ein Kaskade-Tag gesendet, welches mit '88' codiert ist. In einer Single-size-UID darf das erste Byte (uid0) nicht den Wert '88' haben.

Während eines SELECT- oder ANTICOLLISION-Kommandos können maximal 4 Datenbyte an das PCD übertragen werden (siehe auch Bild 3.99). Enthält eine Karte eine Double- oder Triple-Size-Identifikationsnummer, signalisiert sie dies dem Terminal im

Tabelle 3.10 Codierung von SEL (*select code*). Im Select-Code wird angezeigt, in welchem Kaskade-Level sich das Terminal befindet.

b8	b7	b6	b5	b4	b3	b2	b1	Bedeutung
1	0	0	1	0	0	1	1	'93': selektiere Kaskade-Level 1
1	0	0	1	0	1	0	1	'95': selektiere Kaskade-Level 2
1	0	0	1	0	1	1	1	'97': selektiere Kaskade-Level 3
1	0	0	1	alle anderen Werte mit Ausnahme der oben stehenden				RFU

SAK (*Select Acknowledge*) durch ein gesetztes Kaskade-Bit, und sie verbleibt im READY-Status. Das Terminal startet dann das Antikollisionsverfahren erneut, um Byte 5 bis 7 des UID zu ermitteln. Im Falle einer Triple-Size-Identifikationsnummer muss das Antikollisionsverfahren ein drittes Mal durchlaufen werden, um auch noch Byte 8 bis 10 zu ermitteln. Im SELECT-Kommando wird der Karte signalisiert, in welchem Kaskade-Level sich das Terminal befindet, sodass klar ist, welcher Teil der UID abgefragt wird.

Der oben erläuterte Ablauf ist in Bild 3.100 als Flussdiagramm dargestellt.

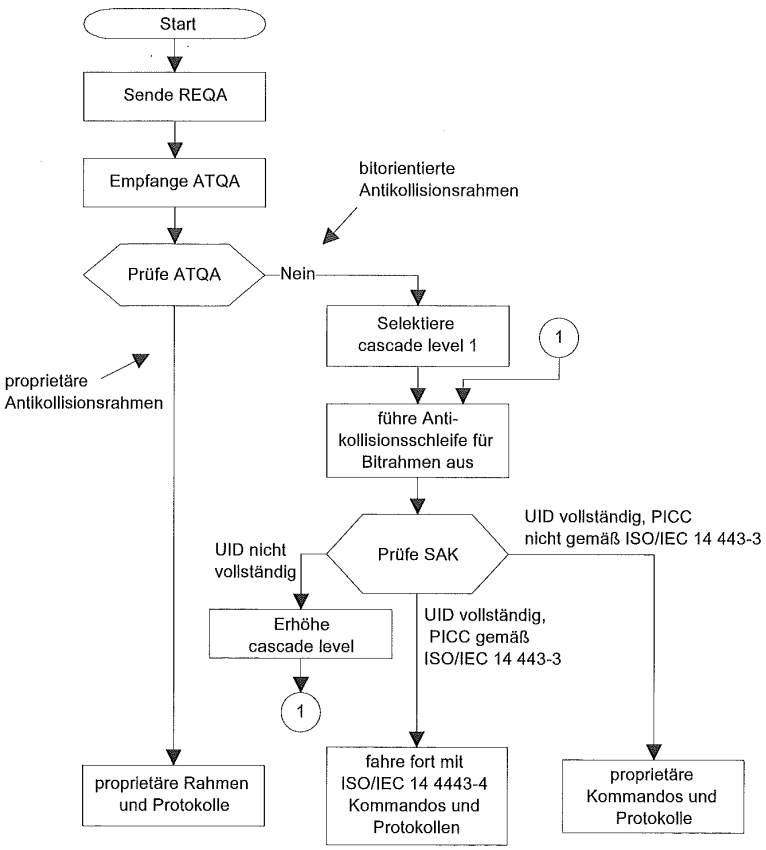


Bild 3.100 Flussdiagramm für die Auswahl einer bestimmten Karte. Nachdem das Terminal ein REQA-Kommando gesendet hat, antworten alle Karten, die sich im IDLE-Zustand befinden, gleichzeitig mit ATQA.

Solange das NVB-Byte keine 40 Bit spezifiziert, handelt es sich um ein ANTICOLLISION-Kommando, und die Karte verbleibt im READY- oder READY*-Zustand. Hat das Terminal die vollständige UID ermittelt, sendet es ein SELECT-Kommando mit NVB = '70', was bedeutet, dass 40 Datenbit spezifiziert sind. Diesem Kommando wird eine

CRC_A-Checksumme angehängt. Die Bildung der Checksumme CRC_A entspricht ISO/IEC 13239. Im informativen Anhang A von ISO/IEC14443-3 ist die Berechnung der Checksumme an einem Beispiel erläutert. Wird die Karte von einem SELECT-Kommando mit ihrer vollständigen Identifikationsnummer angesprochen, wechselt sie vom Zustand READY in den Zustand ACTIVE bzw. vom Zustand READY* in den Zustand ACTIVE* (siehe auch Bild 3.93) und zeigt durch Senden von SAK (*Select Acknowledge*) an, dass die UID vollständig ist. Die Codierung von NVB (*number of valid bits*) und SAK (*select acknowledge*) ist in den folgenden Tabellen dargestellt.

Tabelle 3.11 Codierung von NVB (*number of valid bits*). Die oberen 4 Bit werden „Byte count" genannt und geben die Anzahl vollständiger Datenbytes (einschließlich SEL und NVB) an, die vom Terminal gesendet werden. Die unteren 4 Bit heißen „bit count" und geben die Anzahl der Bits an, die in einem eventuell vorhanden unvollständigen Datenbyte vom Terminal gesendet werden (siehe auch Bild 3.99).

b8	b7	b6	b5	b4	b3	b2	b1	Bedeutung
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 = 4
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

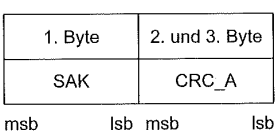


Bild 3.101 Struktur von SAK (*Select acknowledge*). SAK wird von der Karte gesendet, wenn NVB 40 Datenbit anzeigt und diese mit der UID der Karte übereinstimmen.

Tabelle 3.12 Codierung von SAK (*Select acknowledge*).

b8	b7	b6	b5	b4	b3	b2	b1	Bedeutung
x	x	x	x	x	1	x	x	Kaskade Bit gesetzt: UID nicht vollständig
x	x	1	x	x	0	x	x	UID vollständig, PICC gemäß ISO/IEC 14443-4
x	x	0	x	x	0	x	x	UID vollständig, PICC nicht gemäß ISO/IEC 14 443-4

Der Ablauf in der Antikollisionsschleife ist im Bild 3.102 als Flussdiagramm für das Terminal dargestellt. Die Schleife muss in jedem Kaskade-Level durchlaufen werden, wenn die UID dem Terminal nicht bekannt ist.

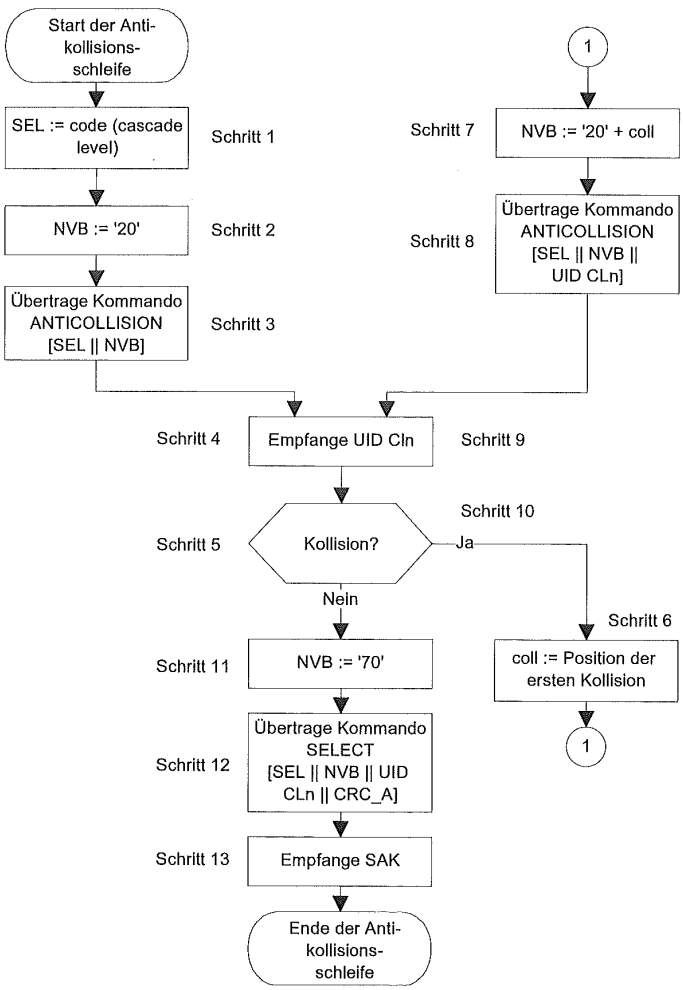


Bild 3.102 Flussdiagramm der Antikollisionsschleife aus Sicht des Terminals. Die Zahlen in den Kreisen entsprechen den unten aufgeführten Schritten des Algorithmus.

In der folgenden Tabelle sind die einzelnen Schritte des Antikollisionsalgorithmus erläutert. Die Schritte entsprechen der Nummerierung in Bild 3.102.

Tabelle 3.13 Schritte der Antikollisionsschleife.

Schritt 1	Das Terminal bestimmt den Select Code (SEL) für den Kaskaden-Level.
Schritt 2	Das Terminal setzt das NVB Byte (*number of valid bits*) auf '20'. Hierdurch wird angezeigt, dass das Terminal keinen Teil der UID senden wird. Daraus folgt, dass alle Karten, die sich im Aktionsfeld des Terminals befinden, ihre komplette UID CLn (Teil der UID für Kaskade-Level n) senden.
Schritt 3	Das Terminal sendet das ANTICOLLISION-Kommando mit SEL und NVB.
Schritt 4	Alle Karten in Reichweite des Terminals antworten mit ihrer vollständigen UID CLn.
Schritt 5	Wenn sich mehrere Karten mit unterschiedlichen Nummern im Feld befinden, wird eine Kollision auftreten. Falls keine Kollision auftritt, werden die Schritte 6 bis 10 übersprungen.
Schritt 6	Das Terminal stellt die Bitposition der ersten Kollision fest.
Schritt 7	Das Terminal gibt NVB den Wert, der die Anzahl der gültigen Bits von UID CLn angibt. Die gültigen Bits sind jener Teil der UID CLn, der empfangen wurde, bevor eine Kollision auftrat, ergänzt um eine °0° oder °1°.
Schritt 8	Das Terminal sendet ein ANTICOLLISION-Kommando mit SEL, NVB und den gültigen Bit.
Schritt 9	Nur die Karten, bei denen der entsprechende Teil der UID CLn den vom Terminal gesendeten gültigen Bits entsprechen, senden die übrigen Bits ihrer UID CLn.
Schritt 10	Falls wieder eine Kollision auftritt, werden die Schritte 6 bis 9 wiederholt.
Schritt 11	Wenn keine Kollision mehr auftritt, setzt das Terminal NVB auf '70'. Das bedeutet, dass das Terminal die vollständige UID CLn senden wird.
Schritt 12	Das Terminal sendet ein SELECT-Kommando mit SEL, NVB und der vollständigen UID CLn, gefolgt von der Checksumme CRC_A.
Schritt 13	Die Karte, deren Nummer mit UID CLn übereinstimmt, antwortet mit SAK.
Schritt 14	Wenn die UID vollständig ist, sendet die Karte ein SAK mit gelöschtem Kaskade-Bit und wechselt vom READY- in den ACTIVE-Status bzw. vom READY*- in den ACTIVE*-Status.
Schritt 15	Das Terminal überprüft, ob das Kaskade-Bit gesetzt ist, um zu entscheiden, ob weitere Antikollisionsschleifen mit höherem Kaskade-Level folgen müssen.

Zur Veranschaulichung der Selektion dient folgendes Beispiel in Bild 3.103, in dem sich zwei Karten im Feld des Terminals befinden. Die Karte 1 (PICC 1) in unserem Beispiel hat eine UID einfacher Größe mit dem Wert '10' für uid0, die Karte 2 (PICC 2) besitzt eine UID mit doppelter Größe.

Die Selektion wird vom Terminal durch Senden eines REQA-Kommandos begonnen. Auf dieses Kommando antworten alle Karten, die sich in Reichweite des Terminals befinden. In unserem Beispiel antwortet PICC 1 mit einem ATQA, in dem durch das gesetzte Bit b1 die „Bit frame anticollision" signalisiert und durch die nicht gesetzten Bits b7 und b8 die UID einfacher Größe angezeigt wird. PICC 2 zeigt in seinem ATQA ebenfalls durch ein gesetztes Bit b1 die „bit frame anticollision" an. Durch das gesetzte Bit b7 wird signalisiert, dass PICC 2 eine UID von doppelter Größe hat.

Im nächsten Schritt beginnt die eigentliche Antikollisionsschleife mit dem Kaskade-Level 1. Das Terminal sendet ein ANTICOLLISION-Kommando mit dem Select-Code '93', wodurch ein Antikollisionsrahmen mit Kaskade-Level 1 angezeigt wird. Der Wert '20' für NVB (number of valid bits) bedeutet, dass das Terminal keinen Teil von UID Kaskade-Level 1 überträgt. Alle Karten im Arbeitsbereich des Terminals antworten infolgedessen mit ihrer vollständigen UID Kaskade-Level 1. Die erste Bitkollision gibt es bei Bit b4. Das Terminal erkennt diese Bitkollision und sendet erneut ein ANTICOLLISION-Kommando welches diesmal die ersten drei vor der ersten Kollision fehlerfrei empfangenen Bit von UID CL1 enthält gefolgt von einer °1°. Infolgedessen gibt das Terminal dem NVB den Wert '24'. Die ersten 4 Bit stimmen nun mit den ersten vier Bit der UID CL1 von der Kar-

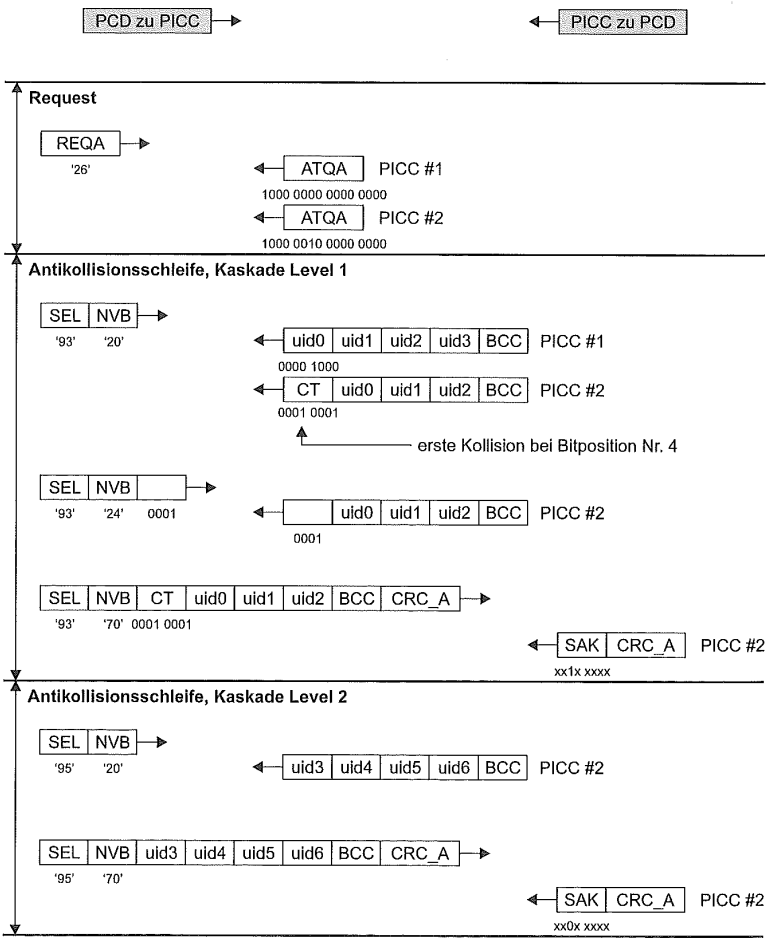


Bild 3.103 Beispiel für eine Initialisierungs- und Antikollisionssequenz für Typ A. Aus Vereinfachungsgründen zeigt das Bild nur die wesentlichen Elemente der Kommunikation.

te 2 überein, nicht jedoch mit den ersten vier Bit der UID CL1 von der Karte 1. Deshalb antwortet nur die Karte 2 und zwar mit den fehlenden 36 Bit ihrer UID CL1. Da das Terminal nun die UID CL1 von Karte 2 vollständig kennt, sendet es ein SELECT-Kommando für die Karte 2. Hierauf antwortet die Karte 2 mit einem SAK (Select Acknowledge), bei dem das Kaskade-Bit b3 gesetzt ist. Hieran erkennt das Terminal, dass die UID noch nicht komplett ist, und erhöht deshalb den Kaskade-Level.

Das Terminal sendet erneut ein ANTICOLLISION-Kommando, in welchem der Select-Code SEL die „bit frame anticollision" und Kaskade-Level 2 anzeigt. Das NVB-Byte ist auf '20' gesetzt, um von der Karte 2 die komplette UID CL2 abzurufen. Die Karte 2 antwortet hierauf mit allen 40 Bit ihrer UID CL2. Das Terminal kann nun ein SELECT-Kommando mit allen Bits der UID CL2 senden, worauf die Karte 2 mit SAK (*select acknowledge*) antwortet, bei dem das Kaskade-Bit b3 nicht gesetzt ist, wodurch angezeigt wird, dass seine UID komplett ist. Die Karte 2 wechselt nun vom READY-Status in den ACTIVE-Status und kann jetzt Kommandos höherer Protokollschichten empfangen.

3.6.3.3.2 Initialisierung und Antikollision Typ B

Proximity-Karten vom Typ B verwenden zur Selektion ein dynamisches Slotted-ALOHA-Verfahren. Bei diesem Verfahren senden Karten, die sich im Arbeitsbereich des Terminals befinden, in festgelegten Zeitschlitzen (*slots*) ihre Daten an das Terminal. Die Auswahl der Zeitschlitze erfolgt nach dem Zufallsprinzip. Die Anzahl der Zeitschlitze kann dabei vom Terminal gewählt werden. Werden nur wenige Zeitschlitze zur Verfügung gestellt und befinden sich deutlich mehr Karten im Aktionsfeld als Zeitschlitze verfügbar sind ist die Wahrscheinlichkeit, dass eine Karte ihre Daten störungsfrei übertragen kann, gering. Eine störungsfreie Übertragung kommt in der Regel nur dann zustande, wenn ein Zeitschlitz nur durch eine Karte belegt wird oder wenn bei einer Mehrfachbelegung eine Karte sich deutlich näher am Terminal befindet als die anderen und sich deshalb ihre Daten wegen des größeren Signalpegels am Terminal durchsetzen. Eine Erhöhung der Anzahl der Zeitschlitze erhöht zwar die Wahrscheinlichkeit einer ungestörten Übertragung, hat allerdings den Nachteil, dass sich der Zeitbedarf für eine Abfrageschleife vergrößert, da über die Summe aller Zeitschlitze auf mögliche Karten gewartet werden muss, auch wenn sich nur eine Karte im Aktionsfeld des Terminals befindet.

Des leichteren Verständnisses wegen wird zunächst das Funktionsprinzip des Verfahrens an einem vereinfachten Beispiel erläutert, bevor das in der ISO/IEC 14443-3 spezifizierte Verfahren im Detail beschrieben wird.

Das Terminal benötigt für die Durchführung des Algorithmus zwei Kommandos:

- *Request:* Dieses Kommando veranlasst alle Karten im Arbeitsbereich, in einem der folgenden Zeitschlitze ihre Identifikationsnummer zu übertragen. In diesem Beispiel sollen 4 Zeitschlitze zur Verfügung stehen.
- *Select:* Dieses Kommando sendet eine zuvor ermittelte Identifikationsnummer an die Karten im Feld. Die Karte mit der passenden Nummer wird hierdurch aktiviert. Alle Karten mit anderen Nummern bleiben passiv und reagieren auch weiterhin nur auf ein Request-Kommando oder ein Select-Kommando mit der passenden Nummer.

Befindet sich das Terminal im Bereitschaftszustand, sendet es in regelmäßigen Zeitabständen ein Request-Kommando aus. Wir nehmen an, dass sechs Karten gleichzeitig mit den Identifikationsnummern 1 bis 6 in den Arbeitsbereich des Terminals gebracht werden. Alle sechs Karten erkennen gleichzeitig das nächstfolgende Request-Kommando und wählen nach dem Zufallsprinzip einen Zeitschlitz, um ihre Identifikationsnummer an das Terminal zu senden (Bild 3.104). In unserem Beispiel kommt es in den Zeitschlitzen 1 und 3 zu einer Kollision. In den Zeitschlitzen 2 und 4 können die Nummern 2 und 3 störungsfrei übertragen werden. Das Terminal kann nun eine der beiden Karten durch ein Select-Kommando auswählen und ohne weitere Kollision mit der ausgewählten Karte kommunizieren.

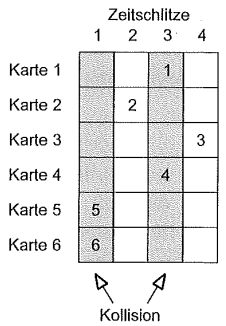


Bild 3.104 Antikollisionsverfahren für ein kontaktloses Kartensystem Typ B. In den Zeitschlitzen 1 und 3 tritt eine Kollision auf, sodass nur die Identifikationsnummern 2 und 3 störungsfrei übertragen werden können. Eine detaillierte Erklärung befindet sich im Text.

Ist die Kommunikation mit der ausgewählten Karte beendet, kann durch erneutes Aussenden des Requestkommandos nach weiteren Karten gesucht werden. Konnte das Terminal nach dem ersten Versuch keine fehlerfreie Identifikationsnummer ermitteln, so wiederholt es das Requestkommando solange, bis es eine gültige Nummer empfängt.

Nachdem das Prinzip des Antikollisionsalgorithmus grundsätzlich erläutert wurde, folgt nun die detaillierte Beschreibung der Kommandos und des Zeitverhaltens von Terminal und Karte, wie sie in ISO/IEC 14443-3 für Typ-B-Karten festgelegt sind. Die Norm definiert einen Satz an Kommandos, der die Implementierung unterschiedlicher Antikollisionsstrategien zulässt. Dem Anwender ist hierdurch ein Freiraum zur Optimierung seines Systems gegeben.

Format und Zeitverhalten von Zeichen (*character*) und Rahmen (*frame*)

Das Terminal und die Karte senden die Datenbytes als Zeichen (*character*), wobei mehrere Zeichen zu Rahmen (*frames*) zusammengefasst werden. Zur Fehlererkennung wird den Datenbytes eines Rahmens ein CRC-Prüfsumme (CRC_B) von 2 Byte Länge angehängt, die sich nach ISO/IEC 13239 berechnet. Im Anhang B von ISO/IEC 14443-3 ist ein Beispiel für die Berechnung von CRC_B angegeben.

Ein Zeichen (*character*) besteht aus einem Startbit (logisch 0), gefolgt von 8 Datenbit mit dem niederwertigsten Bit zuerst und einem Stopbit (logisch 1).

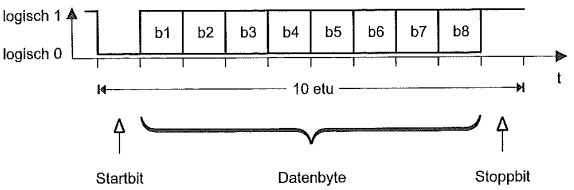


Bild 3.105 Aufbau eines Zeichens.

Zwischen zwei Zeichen gibt es eine Schutzzeit (*Extra Guard Time* – EGT), während der sich Sender und Empfänger auf das nächste Zeichen vorbereiten können. Diese Schutzzeit liegt bei der Übertragung zur Karte zwischen 0 und 57 µs, in der Gegenrichtung zwischen 0 und 19 µs.

Mehrere Zeichen werden bei der Übertragung in beiden Richtungen zu Rahmen zusammengefasst, die mit einem SOF (*start of frame*) beginnen, gefolgt von den zu übertragenden Zeichen, und mit einem EOF (*end of frame*) enden. Das SOF-Zeichen besteht aus einer fallenden Flanke, auf die 10 etu mit einer logischen 0 folgen. Innerhalb der elften etu folgt eine steigende Flanke, nach der für minimal 2 und maximal 3 etu logisch 1 folgt.

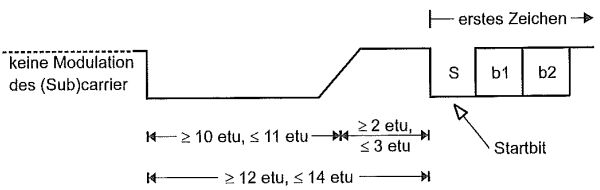


Bild 3.106 Zeitverhalten des SOF (*start of frame*) Zeichens.

Das EOF Signal beginnt ebenfalls mit einer fallenden Flanke, gefolgt von 10 etu mit logisch 0 und einer ansteigenden Flanke innerhalb der elften etu.

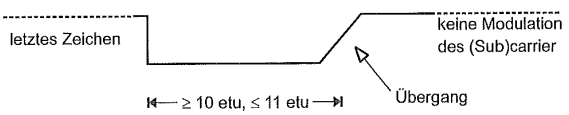


Bild 3.107 Zeitverhalten des EOF (*end of frame*) Zeichens.

Um im Fehlerfall eine Blockade des Übertragungsprotokolls zu verhindern und um der Karte für ihre internen Aktionen definierte Mindest- und Maximalzeiten zu geben, sind die Zeiten zwischen zwei Rahmen, die in entgegengesetzter Richtung übertragen werden, in der Norm festgelegt.

Nachdem die Karte das EOF eines Rahmens, der vom Terminal gesendet wurde, erkannt hat, wartet sie die Guardtime TR0 ab, bevor sie den unmodulierten Hilfsträger generiert. Nach einer weiteren Wartezeit TR1 (*synchronization time*) beginnt die Karte mit der Modulation des Hilfsträgers. Der Minimalwert für TR0 ist $64/f_S$, der Maximalwert während der Antikollisionsschleife (d. h. für ATQB) $256/f_S$. Für alle anderen Rahmen berechnet sich der Maximalwert nach der Formel

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

Der Wert FWI liegt zwischen 0 und 14 und wird dem Terminal im ATQB (*answer to request*, Typ B) mitgeteilt.

Der Minimalwert für die Synchronisationszeit TR1 ist $80/f_S$, der Maximalwert $200/f_S$.

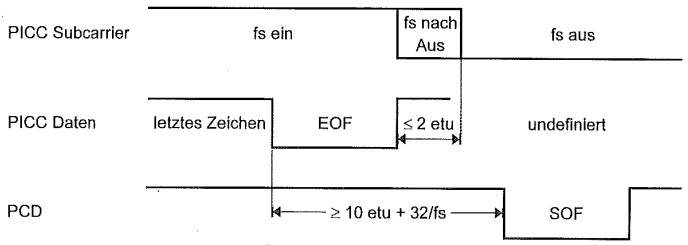


Bild 3.108 Definition der Guardtime TR0 und der Synchronisationszeit TR1.

Hat das Terminal das Ende eines Rahmens, der von der Karte gesendet wurde, erkannt, so wartet es mindestens die Zeit von $32/f_S$ ab, bevor es mit dem Senden eines Rahmens beginnt. Während dieser Wartezeit schaltet die Karte den Hilfsträger ab, und zwar innerhalb von zwei etu nach dem Ende des gesendeten Rahmens.

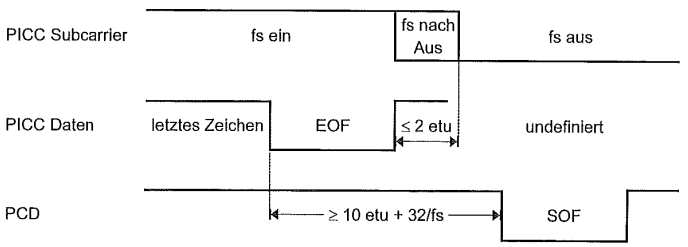


Bild 3.109 Definition der Wartezeit zwischen einem Rahmen, der von der Karte gesendet wurde, und dem darauf folgenden Rahmen, den das Terminal sendet. Das Terminal schaltet den Hilfsträger erst nach dem Ende des EOF-Zeichens ab.

Ablauf der Selektion einer Karte

Gelangt eine Typ-B-Karte in den Arbeitsbereich eines Terminals, wird der Mikroprozessor der Karte mit Energie versorgt und gelangt nach dem Power-on-reset in den IDLE-Zustand. Ebenso wie die Typ-A-Karten müssen die Typ-B-Karten den IDLE-Zustand innerhalb von 5 ms einnehmen, nachdem sie durch das Feld des Terminals ausreichend mit Energie versorgt wurden. Im IDLE-Zustand darf die Karte keine Rahmen senden, sondern sie muss ausschließlich auf den Empfang eines gültigen REQB- oder WUPB-Kommandos warten. Das REQB- bzw. WUPB- Kommando beinhaltet einen Parameter, der die Anzahl der Zeitschlitze angibt, die das Terminal verwendet, sowie einen „Application Family Identifier, AFI", mit dem eine bestimmte Anwendungsgruppe vorgegeben werden kann. Hierdurch wird eine Vorselektion getroffen, da nur jene Karten antworten, deren Anwendungen zur angezeigten „Application Family" passen.

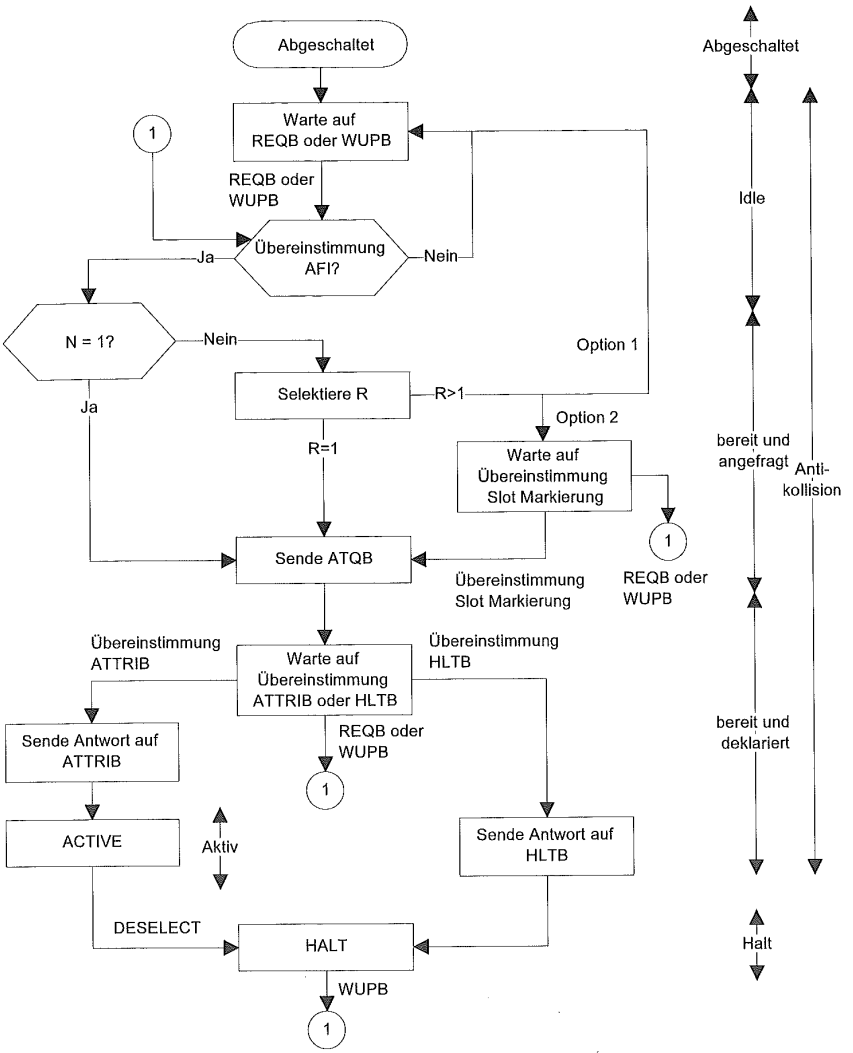


Bild 3.110 Zustandsdiagramm einer Typ-B-Chipkarte nach ISO/IEC 14 443.

REQB/WUPB-Kommando

Das REQB- und das WUPB-Kommando werden vom Terminal gesendet, um festzustellen, ob sich eine Typ-B-Karte in seinem Arbeitsbereich befindet. Das WUPB-Kommando wird speziell dafür verwendet, Karten, die sich im HALT-Zustand befinden, aufzuwecken. Das REQB/WUPB-Kommando besteht aus dem Antikollisions-Prefix (APf) mit dem Wert Ta'05',gefolgt von dem AFI-Byte, einem PARAM-Byte, das die Anzahl der Zeitschlitze angibt, sowie den zwei CRC_B-Bytes.

Tabelle 3.14 Aufbau des REQB/WUPB-Kommandos

APf	AFI	PARAM	CRC_B
1 Byte	1 Byte	1 Byte	2 Byte

Tabelle 3.15 Codierung des „application family identifier, AFI"

AFI höherwertiges Halbbyte	AFI niederwertiges Halbbyte	Bedeutung	Beispiel, Bemerkung
'0'	'0'	alle Familien und Unterfamilien	keine Vorselektion der Anwendung
X	'0'	alle Unterfamilien von Familie X	weitergehende Vorselektion der Anwendung
X	Y	nur die Y Unterfamilie von Familie X	
'0'	Y	nur proprietäre Unterfamilie Y	
'1'	'0', Y	Transport	ÖPNV, Bus, Luftverkehrsgesellschaft, . . .
'2'	'0', Y	Finanzsektor	elektronische Börsen, Banken, Einzelhandel, . . .
'3'	'0', Y	Identifizierung	Zutrittskontrolle, . . .
'4'	'0', Y	Telekommunikation	öffentliches Telefon, GSM, . . .
'5'	'0', Y	Medizin	
'6'	'0', Y	Multimedia	Internet-Dienste, . . .
'7'	'0', Y	Spiele	
'8'	'0', Y	Datenspeicherung	transportable Dateien, . . .
'9' – 'F'	'0', Y	RFU	

Tabelle 3.16 Codierung des PARAM-Bytes. b4 = 0 definiert REQB (alle Karten im IDLE- oder READY-Zustand reagieren auf dieses Kommando). b4 = 1 definiert WUPB (alle Karten im IDLE-, READY- oder HALT-Zustand reagieren auf dieses Kommando).

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

Die Anzahl der Zeitschlitze wird nach Tabelle 3.17 codiert.

Tabelle 3.17 Codierung von N (Number of slots). Die Wahrscheinlichkeit, dass eine Karte im ersten Slot reagiert, ist 1/N. Wenn ein Terminal nur einen einzigen Zeitschlitz benutzt, kann über N die Wahrscheinlichkeit angegeben werden, mit der eine Karte in diesem antwortet.

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

Nachdem eine Karte ein gültiges REQB-Kommando empfangen hat, überprüft sie, ob sie eine der im AFI angezeigte Anwendungen unterstützt. Sofern dies der Fall ist, wertet die Karte das PARAM-Byte aus und erhält so den Wert N, der die Anzahl der zur Verfügung gestellten Slots angibt. Die Karte befindet sich nun im Ready-Requested-Zustand.

Ist N = 1, sendet die Karte ein ATQB (*answer to request*, Typ B) und wechselt in den Ready-Declared-Status.

Ist N > 1, erzeugt die Karte eine Zufallszahl R mit einer Gleichverteilung zwischen 1 und N.

Ist R = 1, sendet die Karte ein ATQB (*answer to request*, Typ B) und wechselt in den Ready-Declared-Status.

Für R > 1, sind in der Norm zwei Optionen vorgesehen, die unterschiedliche Algorithmen unterstützen.

Option 1: Diese Karten unterstützen nicht die Auswahl bestimmter Zeitschlitze. Sie kehren an dieser Stelle in den IDLE-Status zurück. Die Karten durchlaufen diese Schleife so lange, bis zufällig der Wert R = 1 erzeugt wird (*probabilistic approach*) oder das Terminal den Wert für N auf 1 setzt. Bei dieser Option handelt es sich nicht um ein Zeitschlitzverfahren im eigentlichen Sinne, da nur ein einziger Zeitschlitz benutzt wird. Das Verfahren ist einfach zu implementieren und für Systeme, bei denen nur wenige Karten gleichzeitig im Arbeitsbereich des Terminals vorkommen, ausreichend.

Option 2: Diese Karten unterstützen die Auswahl von Zeitschlitzen. Sie warten, bis sie ein Slotmarker-Kommando empfangen, welches die passende Zeitschlitznummer (*slot number* = R) enthält, bevor sie ein ATQB aussenden und in den Ready-Declared-Status wechseln.

Slotmarker-Kommando

Ein Slotmarker-Kommando sendet das Terminal zu Beginn eines jeden Zeitschlitzes aus. Es hat folgendes Format wie in Bild 3.111 dargestellt.

APn	CRC_B
1 Byte	2 Byte

Bild 3.111 Format des Slotmarker-Kommandos.

Die Codierung des „Anticollision Prefix Byte" ist APn = 'n5' wobei n die Nummer des nachfolgenden Slot angibt. Die Slotmarker können in beliebiger Reihenfolge, das heißt, sie müssen nicht unbedingt in aufsteigender Reihenfolge ihrer Nummern gesendet werden.

Tabelle 3.18 Codierung der „slot number" im „anticollision prefix byte" APn.

nnnn	Slot number
0001	2
0010	3
0011	4
.....	.....
1110	15
1111	16

Nach dem Aussenden eines Slotmarker-Kommandos erkennt das Terminal nach Ablauf von $TR0_{max} = 256/f_S$, ob eine Karte mit der Übertragung ihres ATQB begonnen hat. Ist dies nicht der Fall, kann das Terminal sofort den nächsten Slotmarker senden.

ATQB, (*Answer To Request*, Typ B)
Das Format von ATQB, welches als Antwort auf REQB/WUPB- und Slotmarker-Kommandos gesendet wird, ist in Bild 3.112 dargestellt.

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

Bild 3.112 Aufbau eines ATQB (*answer to request*, Typ B).

Das ATQB enthält Informationen über wichtige Parameter der Chipkarte, die das Terminal benötigt, um die Karte zu selektieren.

Die „PUPI" (*Pseudo-Unique PICC Identifier*) ist die Identifikationsnummer der PICC für die Antikollisionsschleife. Diese Nummer kann eine fest mit der Karte verbundene Nummer sein. Sie kann aber auch eine von der Karte erzeugte Zufallszahl sein, die nach jedem Power-On-Reset neu gebildet wird.

Im Feld Application Data stehen Informationen über die in der Karte installierten Applikationen. Diese Informationen ermöglichen es dem Terminal, die gewünschte Karte auszuwählen, falls sich mehrere Karten im Arbeitsbereich befinden. Die Definition der Applikationsdaten hängt von dem ADC-(*Applikation Data Coding*) Feld in der „Protocol Info" ab. Dieses definiert, ob die „CRC_B compressing method" wie im Folgenden beschrieben angewendet wird, oder ob eine proprietäre Codierung benutzt wird.

Wird die „CRC_B compressing method" verwendet, ist das „Application Data field" folgendermaßen aufgebaut:

AFI	CRC_B(AID)	Numbers of Applications
1 Byte	2 Byte	1 Byte

Bild 3.113 Aufbau von Application Data. Die Codierung von AFI ist in Tabelle 3.15 angegeben. Für Multiapplikationskarten gibt AFI die Familie der Applikationen an. CRC_B(AID) ist die Prüfsumme CRC_B des Application Identifier AID nach ISO/IEC 7816-5 einer Applikation in der Karte, die der AFI entspricht, die im REQB/WUPB Kommando gesendet wurde. Numbers of Applications gibt die Anzahl weiterer vorhandener Applikationen an. Das höchstwertige Halbbyte enthält die Zahl der Applikationen, die der AFI entsprechen, wobei '0' keine Applikation und 'F' 15 oder mehr Applikationen bedeutet. Das niederwertige Halbbyte enthält die Gesamtzahl der Applikationen in der Karte mit der Bedeutung '0' keine Applikation und 'F' 15 oder mehr Applikationen.

Das Feld „Protocol Info" zeigt wichtige Parameter an, die von der Karte unterstützt werden. Diese Parameter ermöglichen es dem Terminal, sich für das folgenden Applikationsprotokoll optimal auf die Leistungsfähigkeit der Karte einstellen zu können oder sich auf Karten einzustellen, die nicht alle Anforderung der Norm erfüllen.

1. Byte	2. Byte	3. Byte			
Bit_Rate_capability	Max_Frame_Size	Protocol_Type	FWI	ADC	FO
8 bit	4 bit	4 bit	4 bit	2 bit	2 bit

Bild 3.114 Aufbau des „Protocol Info"-Feldes.

In den folgenden Tabellen ist die Codierung der einzelnen Felder von „Protocol Info" angegeben.

Tabelle 3.19 Von der Karte unterstützte Rahmenoptionen (*Frame Option* – FO).

b2	b1	Bedeutung
1	x	NAD wird von der PICC unterstützt.
x	1	CID wird von der PICC unterstützt.

Tabelle 3.20 Unterstützung von „Application Data Coding" durch die Karte (ADC).

b4	b3	Bedeutung
0	0	Proprietäre Applikation
0	1	Applikation ist codiert wie oben beschrieben.

Der Parameter FWI (*Frame Waiting time Integer*) gibt die Zeit an, die die Karte maximal benötigt, um mit dem Senden einer Antwort zu beginnen, nachdem die Karte ein Kommando vom Terminal vollständig empfangen hat. Antwortet eine Karte nicht innerhalb dieser Zeit, so kann das Terminal davon ausgehen, dass die Kommunikation mit dieser Karte unterbrochen ist.

Die „frame waiting time" FWT wird nach folgender Formel berechnet:

$$FWT = (256 \times 16/f_C) \times 2^{FWI}.$$

Der Wert von FWI liegt zwischen 0 und 14, der Wert 15 ist für zukünftige Verwendung reserviert (RFU). Mit dieser Formel ergeben sich für den Minimal- und Maximalwert der Rahmenwartezeit folgende Werte:

Mit FWI = 0 ergibt sich der Minimalwert von $FWT_{min} \sim 302$ µs,

Mit FWI = 14 ergibt sich der Maximalwert von FWT_{max} von 4949 ms.

Im Feld „Protocol Type" wird angezeigt, ob die Karte das Übertragungsprotokoll von ISO/IEC 14443-4 unterstützt.

Tabelle 3.21 Von der Karte unterstützte Protokolle. Alle anderen Werte sind für zukünftige Verwendung reserviert (RFU).

b4	b3	b2	b1	Bedeutung
0	0	0	1	PICC unterstützt ISO/IEC 14 443-4
0	0	0	0	PICC unterstützt ISO/IEC 14 443-4 nicht

Im Feld „Max_Frame_Size" zeigt die Karte die maximale Größe eines Rahmens an, den sie empfangen kann. Diese wird durch die Größe des Empfangsbuffers im RAM der Karte begrenzt. Bei kostengünstigen Chips ist der RAM in der Regel knapp bemessen, sodass nur kleine Rahmen empfangen werden können.

Tabelle 3.22 Maximal von der Karte verarbeitbare Rahmengröße.

Max_Frame_Size Code in ATQB	0	1	2	3	4	5	6	7	8	9-F
Maximale Rahmengröße in Byte	16	24	32	40	48	64	96	128	256	RFU>256

Im Feld „Bit_Rate_capability" zeigt die Karte an, welche Übertragungsgeschwindigkeiten sie unterstützt. Dies gibt dem Terminal die Möglichkeit, die Leistungsfähigkeit der Karte optimal zu nutzen, indem es seine Datenübertragung entsprechend anpasst.

Tabelle 3.23 Von der Karte unterstützte Bitraten. Alle anderen Werte sind für zukünftige Verwendung reserviert (RFU).

b8	b7	b6	b5	b4	b3	b2	b1	Bedeutung
0	0	0	0	0	0	0	0	PICC unterstützt nur 106 kBit/s in beiden Richtungen.
1	X	X	X	0	X	X	X	Dieselbe Bitrate in beiden Richtungen.
X	X	X	1	0	X	X	X	PICC to PCD, 1 etu = $64/f_C$, unterstützte Bitrate ist 212 kBit/s.
X	X	1	X	0	X	X	X	PICC to PCD, 1 etu = $32/f_C$, unterstützte Bitrate ist 424 kBit/s.
X	1	X	X	0	X	X	X	PICC to PCD, 1 etu = $16/f_C$, unterstützte Bitrate ist 847 kBit/s.
X	X	X	X	0	X	X	1	PCD to PICC, 1 etu = $64/f_C$, unterstützte Bitrate ist 212 kBit/s.
X	X	X	X	0	X	1	X	PCD to PICC, 1 etu = $32/f_C$, unterstützte Bitrate ist 424 kBit/s.
X	X	X	X	0	1	X	X	PCD to PICC, 1 etu = $16/f_C$, unterstützte Bitrate ist 847 kBit/s.

Wie bereits gesagt, wechselt die Karte in den READY-DECLARED-Zustand (siehe auch Bild 3.110), nachdem sie ihr ATQB ausgesandt hat. In diesem Zustand reagiert die Karte nur auf die Kommandos REQB/WUPB. ATTRIB und HLTB.

Auf den Empfang eines REQB/WUPB-Kommandos reagiert die Karte in derselben Weise wie im IDLE-Zustand.

Erkennt sie ein gültiges ATRIB-Kommando, in welchem die PUPI mit der PUPI der Karte übereinstimmt, sendet die Karte einen „Answer to ATTRIB"-Rahmen und wechselt in den Zustand ACTIVE. Stimmen die PUPI nicht überein, bleibt die Karte im Zustand READY-DECLARED und wartet weiter auf ein ATRIB-Kommando mit der richtigen PUPI.

Auf ein passendes HALTB-Kommando (mit der richtigen PUPI) reagiert die Karte mit einem „Answer to HLTB" und wechselt in den Zustand HALT.

Im Zustand ACTIVE hat die Karte einen Card Identifier (CID), der ihr im ATRRIB-Kommando zugewiesen wurde. Sie befindet sich damit in einer höheren Protokollschicht und horcht auf passende Applikationskommandos mit der richtigen CID und korrektem CRC_B.

Spezielle Kommandos dieser höheren Protokollschicht können die Karte in die Zustände IDLE oder HALT versetzen. Im Zustand ACTIVE darf die Karte nicht auf die Kommandos REQB/WUPB, Slot-MARKER oder ATTRIB antworten.

Im Zustand HALT ist die Karte passiv und kann nur durch ein gültiges WUPB-Kommando mit der passenden PUPI in den Zustand IDLE rückversetzt werden.

Aufbau und Codierung des Kommandos ATTRIB

Das Kommando ATTRIB wird vom Terminal an die Karte gesendet und enthält Informationen, die benötigt werden, um eine Karte zu selektieren. Darüber hinaus enthält es Informationen über die Parameter, die das Terminal für die folgende Kommunikation unterstützt und die von der Karte für eine fehlerfreie Kommunikation benötigt werden. Hierzu gehören z.B. der Minimalwert für die Guardtime TR0, der Minimalwert für die Synchronisationszeit TR1, die Möglichkeit der Unterdrückung von SOF und/oder EOF durch die Karte zur Beschleunigung der Kommunikation, die maximale Rahmengröße sowie die Auswahl der optimalen Bitrate.

'1D'	Identifier (PUPI)	Param 1	Param 2	Param 3	Param 4	Higher Layer-Inf	CRC_B
1 BYTE	4 Byte	1 Byte	1 Byte	1 Byte	1 Byte	0 oder mehr Byte	2 Byte

Bild 3.115 Aufbau des Kommandos ATTRIB. „Identifier" enthält dabei den Wert der PUPI, die von der Karte im ATQB übermittelt wurde. Die Struktur von Param 1 ist in Bild 3.116 beschrieben.

b8	B7	b6	b5	b4	b3	b2	b1
Minimum TR0		Minimum TR1		EOF	SOF	RFU	

Bild 3.116 Die Struktur von Param 1.

Codierung von Minimum TR0: Der Wert von Minimum TR0 definiert die Zeit, die eine Karte mindestens warten muss, bevor sie auf ein vom Terminal empfangenes Kommando antwortet. Diese Zeit wird vom Terminal benötigt, um vom Sende- in den Empfangsmodus umzuschalten. Die Dauer hängt von der Leistungsfähigkeit des Terminals ab.

Tabelle 3.24 Codierung von Minimum TR0.

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

Minimum TR1 gibt der Karte die minimale Verzögerungszeit zwischen dem Start des Hilfsträgers und der Datenübertragung (siehe auch Bild 3.108). Das Terminal benötigt diese Zeit zur Synchronisation mit der Karte.

Tabelle 3.25 Codierung von Minimum TR1.

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

Bit 3 und 4 zeigen dem Terminal an, ob die Karte eine Unterdrückung von EOF und/oder SOF von der Karte zum Terminal unterstützt, wodurch der Kommunikationsaufwand reduziert werden kann. Diese Funktion ist für die Karte optional.

Tabelle 3.26 SOF-Handhabung.

b3	SOF erforderlich
0	ja
1	nein

Tabelle 3.27 EOF-Handhabung.

b4	EOF erforderlich
0	ja
1	nein

Der Parameter 2 zeigt in seinem niederwertigen Halbbyte (b4 bis b1) die maximale Rahmengröße an, die vom Terminal empfangen werden kann. Das höherwertige Halbbyte wird zur Auswahl der Bitraten in beiden Richtungen benutzt. Das Terminal kann diese Auswahl treffen, da es bereits aus dem ATQB die von der Karte unterstützten Bitraten kennt.

Tabelle 3.28 Codierung von b4 bis b1 in Parameter 2 zur Definition der maximalen Rahmengröße.

Max_Frame_Size Code in ATTRIB	0	1	2	3	4	5	6	7	8	9-F
Maximale Rahmengröße in Byte	16	24	32	40	48	64	96	128	256	RFU>256

Tabelle 3.29 Codierung von b8 bis b5 in Parameter 2 zur Auswahl der Bitrate.

b8	b7	b6	b5	Bedeutung
0	0	x	x	PICC to PCD, 1 etu = 128/f_C, Bitrate ist 106 kBit/s
0	1	x	x	PICC to PCD, 1 etu = 64/fc, Bitrate ist 212 kBit/s
1	0	x	x	PICC to PCD, 1 etu = 32/f_C, Bitrate ist 424 kBit/s
1	1	x	x	PICC to PCD, 1 etu = 16/f_C, Bitrate ist 847 kBit/s
x	x	0	0	PCD to PICC, 1 etu = 128/f_C, Bitrate ist 106 kBit/s
x	x	0	1	PCD to PICC, 1 etu = 64/f_C, Bitrate ist 212 kBit/s
x	x	1	0	PCD to PICC, 1 etu = 32/f_C, Bitrate ist 424 kBit/s
x	x	1	1	PCD to PICC, 1 etu = 16/f_C, Bitrate ist 847 kBit/s

Im Parameter 3 wird das niederwertige Halbbyte für die Bestätigung des Protokolltyps benutzt. Die Codierung entspricht der Tabelle 3.28. Das höherwertige Halbbyte ist auf '0' gesetzt. Alle anderen Werte sind für zukünftige Verwendung reserviert.

Der Parameter 4 besteht ebenfalls aus zwei Teilen. Das niederwertige Halbbyte heißt Card Identifier (CID) und definiert die logische Nummer der angesprochenen Karte im Bereich von 0 bis 14. Der Wert 15 ist für zukünftige Verwendung reserviert. Der Card Identifier wird durch das Terminal spezifiziert und ist einzigartig für jede aktive Karte. Wenn die Karte den CID nicht unterstützt, wird der Wert '0' verwendet. Das höherwertige Halbbyte ist auf '0' gesetzt. Alle anderen Werte sind für zukünftige Verwendung reserviert.

Das „Higher layer INF"-Datenfeld kann für die Übertragung jeglicher Kommandos aus höheren Schichten benutzt werden. Für die Karte ist die Bearbeitung dieser Kommandos optional.

Antwort auf das Kommando ATTRIB

Die Karte antwortet auf jedes gültige ATTRIB-Kommando (korrekte PUPI und CRC_B) in folgender Form:

1. Byte		**2. – n Byte**	
MBLI	CID	Higher layer response	CRC_B
1 Byte		Optional 0 oder mehr Byte	2 Byte

Bild 3.117 Aufbau der Antwort auf ein ATTRIB-Kommando.

An einer gültigen Antwort (dieselbe CID und korrektes CRC_B) auf ein ATTRIB-Kommando erkennt das Terminal, dass die Selektion der Karte erfolgreich war.

Das niederwertige Halbbyte des ersten Bytes (b4 bis b1) enthalten die CID. Das höherwertige Halbbyte (b8 bis b5) wird als Maximum-Buffer-Längen-Index MBLI bezeichnet. Mit dem MBLI teilt die Karte dem Terminal die Grenze seines Eingangsbuffers mit. Damit kann das Terminal verhindern, dass der Eingangsbuffer der Karte beim Empfang zu vieler verketteter Rahmen überläuft.

Ist MBLI auf 0 gesetzt, gibt die Karte keine Information über ihre interne Buffergröße.

Für MBLI > 0 berechnet sich die maximale interne Bufferlänge (MBL) nach folgender Formel:

$$MBL = (\text{PICC Maximum Frame Size}) \times 2^{(MBL-1)}$$

Die maximale Rahmengröße hatte die Karte bereits in ihrem ATQB an das Terminal gesendet. Das Terminal muss beim Senden verketteter Rahmen darauf achten, dass die kumulierte Länge niemals größer als MBL wird.

Das Kommando HLTB

Das Kommando HLTB dient dazu, eine Karte in den Zustand HALT zu versetzen, sodass es nicht mehr auf REQB-Kommandos reagiert. Nach der Beantwortung dieses Kommandos ignoriert die Karte alle Kommandos mit Ausnahme des WUPB-Kommandos.

'50'	**Identifier**	**CRC_B**
1 Byte	4 Byte	2 Byte

Bild 3.118 Aufbau eines HALTB-Kommandos.

Der Identifier enthält den PUPI der Karte, die in den HALT-Zustand versetzt werden soll.

Die Karte antwortet auf ein gültiges HALT-Komando folgendermaßen:

'00'	**CRC_B**
1 Byte	2 Byte

Bild 3.119 Antwort auf ein HALTB-Kommando.

Beispiel für eine Antikollisionssequenz mit drei Typ-B-Karten

Die Norm lässt dem Entwickler den Freiraum für die Implementierung unterschiedlicher Antikollisionsstrategien. Dies entspricht dem Grundgedanken der Normung, Interoperabilität zu ermöglichen, gleichzeitig aber möglichst viel Freiraum für die Implementierung zu lassen, um den technischen Fortschritt nicht zu behindern. Das folgende Beispiel, welches auch im Anhang der Norm enthalten ist, dient deshalb dazu, die im vorangehenden Kapitel definierten und beschriebenen Abläufe und Kommandos zu veranschaulichen. Es erhebt nicht den Anspruch auf eine technisch vorbildliche Implementierung.

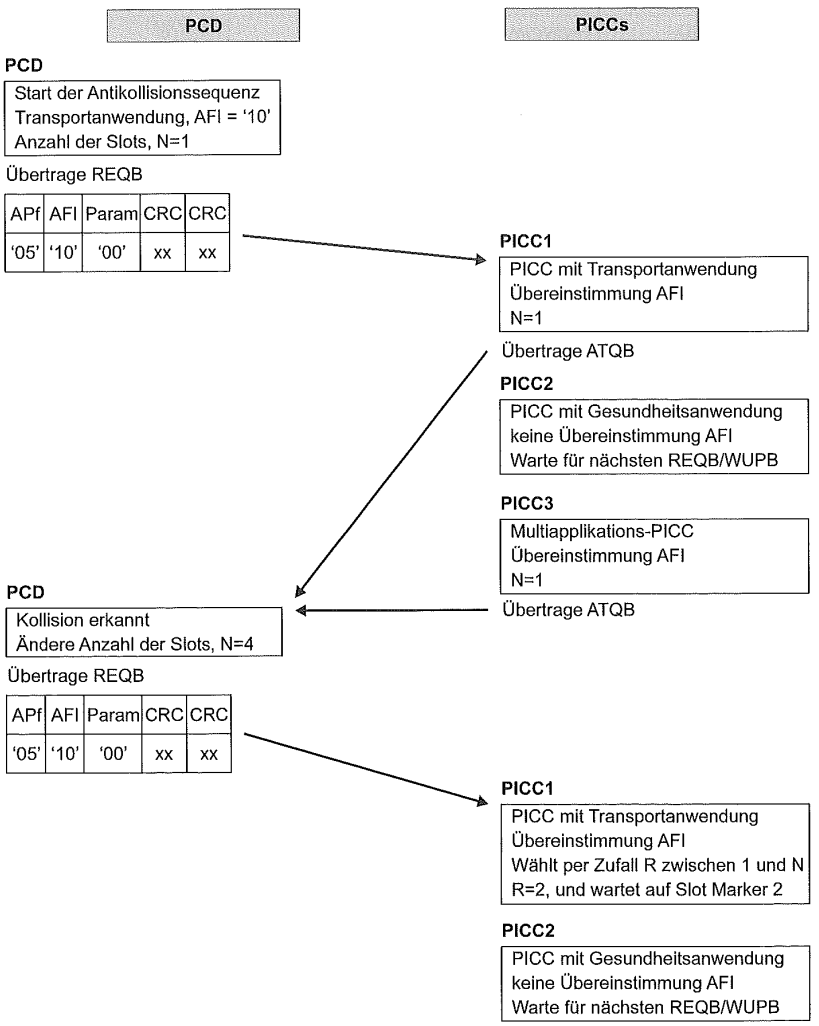


Bild 3.120 Beispiel für eine Antikollisionssequenz mit drei Typ-B-Karten (Teil 1).

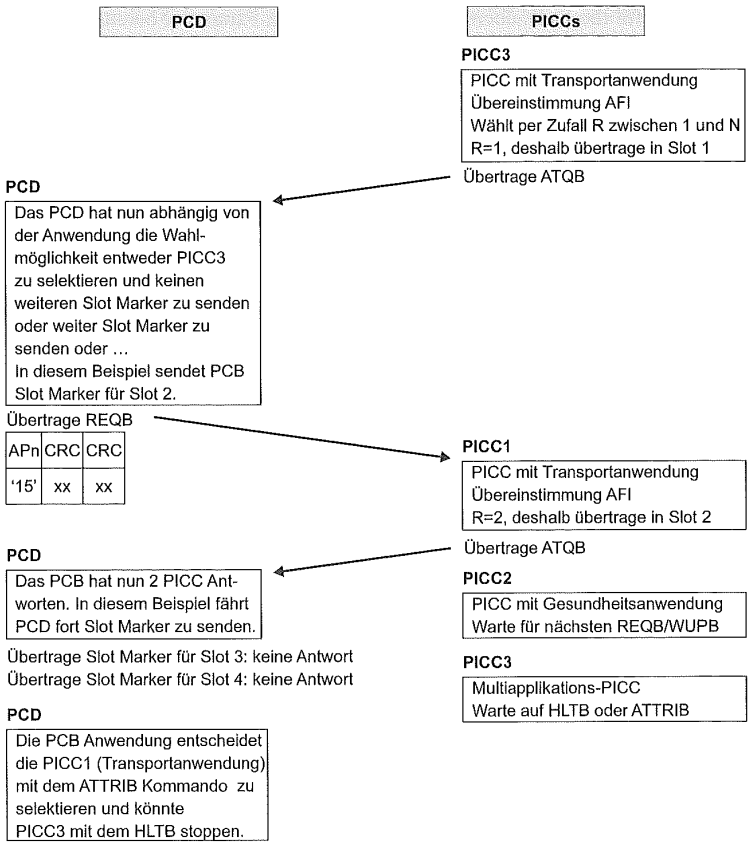


Bild 3.121 Beispiel für eine Antikollisionssequenz mit drei Typ-B-Karten (Teil 2).

3.6.3.4 Übertragungsprotokoll (ISO/IEC 14443-4)

Im Teil 4 der ISO/IEC 14443 wird ein Halbduplexblockprotokoll definiert, das auf die speziellen Anforderungen eines kontaktlosen Systems zugeschnitten ist. Es lehnt sich weitgehend an das in ISO/IEC 7816-3 definierte T = 1 Protokoll an, welches international weite Verbreitung gefunden hat. Hierdurch wird die Implementierung von Dual-Interface-Karten erleichtert, die ja auch ein Übertragungsprotokoll für kontaktbehaftete Karten unterstützen müssen.

Für Karten vom Typ-A definiert die Norm eine Aktivierungssequenz, die vor dem Start des eigentlichen Protokolls durchlaufen werden muss. In dieser Sequenz werden Parameter für die folgende Datenübertragung zwischen Terminal und Karte ausgetauscht und festgelegt. Hierzu gehören zum Beispiel die Bitrate in beiden Richtungen und die geforderten Wartezeiten zwischen zwei Rahmen.

Karten vom Typ-B benötigen keine besondere Aktivierungssequenz, sondern können nach der Selektion unmittelbar mit dem eigentlichen Übertragungsprotokoll starten. Bei ihnen

wurden die für die Datenübertragung erforderlichen Parameter bereits bei der Initialisierung und Selektion ausgetauscht und festgelegt, wie im vorangehenden Kapitel beschrieben ist.

3.6.3.4.1 Protokoll-Aktivierung bei Karten vom Typ A

Nachdem eine PICC vom Typ A erfolgreich selektiert wurde (siehe Abschn. 3.6.3.3.1), führt das Terminal eine Aktivierungssequenz durch, die in Bild 3.122 als Flussdiagramm dargestellt ist.

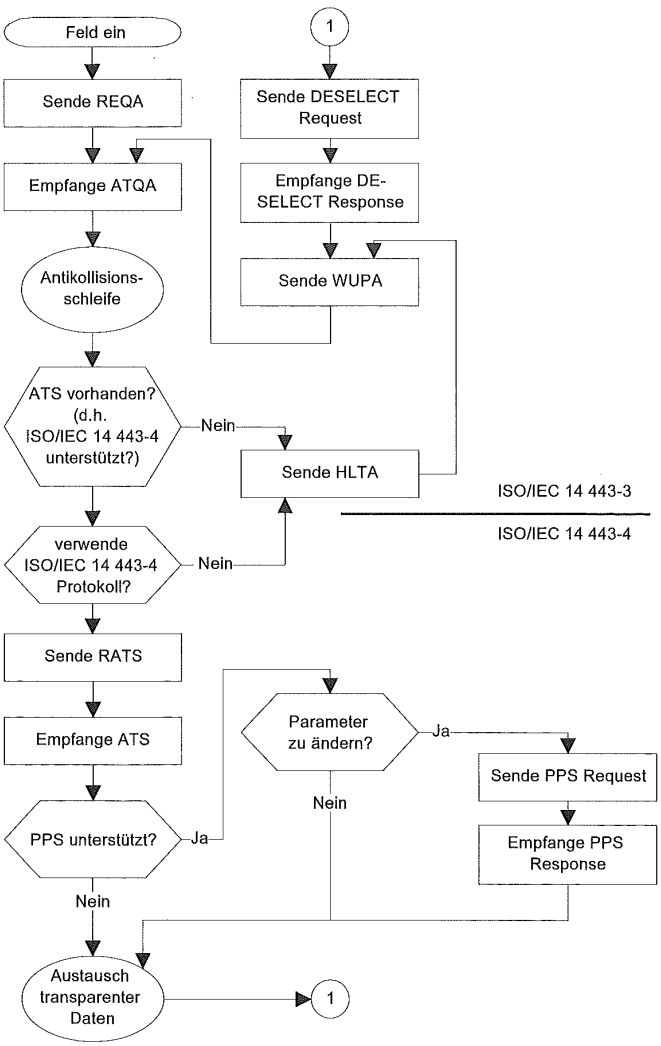


Bild 3.122 Aktivierung einer Karte vom Typ A durch das Terminal.

Das Terminal kann am SAK (*select acknowledge*), das am Ende der Antikollisionsschleife von der Karte gesendet wurde, erkennen, ob die Karte das Standardübertragungsprotokoll unterstützt. Ist dies nicht der Fall, versetzt das Terminal die Karte mit einem HLTA-Kommando in den HALT-Zustand. Wird das Protokoll nach ISO/IEC 14443-4 unterstützt, sendet das Terminal ein RATS-Kommando (*Request for Answer To Select*) an die Karte. Das RATS-Kommando und die von der Karte zurückgeschickte ATS (*Answer To Select*) dienen zum Austausch von Informationen und Parametern, welche die unterstützten Optionen der Karte und des Terminals für die Datenübertragung bestimmen. Danach können die veränderbaren Parameter mithilfe der Protokoll- und Parameterselektion PPS so ausgewählt werden, dass die Leistungsfähigkeit von Karte und Terminal optimal genutzt werden. Um auch kostengünstige, technisch einfache Implementierungen zu ermöglichen, sind für die veränderbaren Parameter Default-Werte definiert. Im einfachsten Fall unterstützt eine Karte nur diese Default-Werte. Die PPS-Sequenz entfällt dann, und das Terminal kann unmittelbar nach Empfang des ATS mit dem Austausch von Daten mithilfe des Blockprotokolls beginnen.

Request for Answer to Select (RATS)

Das RATS-Kommando enthält ein Parameter-Byte, in dem die maximale Rahmengröße angegeben wird, die das Terminal empfangen kann (FSDI – *Frame Size for proximity coupling device*) sowie den CID (*Card Identifier*), den die Karte für die Dauer ihres aktiven Zustandes zugeteilt bekommt. Die Karte benutzt von nun an diesen CID als ihren logischen Identifikator.

'E0'	Parameter	CRC_A
1 Byte	1 Byte	2 Byte

Bild 3.123 Aufbau des RATS-Kommandos (*Request for Answer To Select*).

b8	b7	b6	b5	b4	b3	b2	b1
FSDI				CID			

Bild 3.124 Codierung des Parameter-Byte von RATS. CID definiert die logische Nummer der adressierten Karte im Bereich von 0 bis 14. Der Wert 15 ist für zukünftige Verwendung reserviert. FSDI codiert die maximale Größe eines Rahmens, die das Terminal empfangen kann (FSD).

Tabelle 3.30 Codierung von b8 bis b5 des Parameters im RATS-Kommando

FSDI	0	1	2	3	4	5	6	7	8	9-F
FSD in Byte	16	24	32	40	48	64	96	128	256	RFU>256

Answer to Select (ATS)

Eine Typ-A-Karte antwortet auf ein RATS-Kommando mit einem ATS (*Answer To Select*). Die ATS enthält Informationen darüber, welche Auswahl an Parametern von der Karte unterstützt werden. Hierzu gehören die:

- Maximale Rahmengröße,
- Bitraten in beide Übertragungsrichtungen, die von der Karte unterstützt werden,
- Wartezeit zwischen zwei Rahmen,

- Spezifische Rahmenschutzzeit,
- Unterstützung von NAD und CID.

Für Karten, die keine Auswahl an Parametern bieten, sind Default-Werte spezifiziert. Im kürzesten Fall, wenn nur die Default-Werte unterstützt werden, besteht der ATS nur aus dem Längenbyte und den CRC-Bytes.

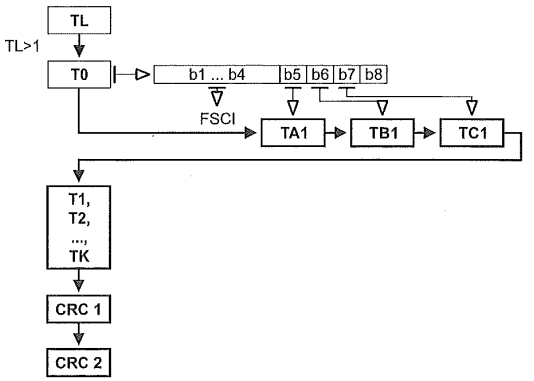


Bild 3.125 Der Aufbau des ATS (*Answer to Select*).

Tabelle 3.31 Die Datenelemente des ATS und ihre Bezeichnung nach ISO/IEC 14 443-4

Datenelement	Bezeichnung
TL	Length Byte
T0	Format Byte
TA1, TB1, TC1	Interface Bytes
T1, T2, . . .Tk	Historical Bytes
CRC1, CRC2	Cyclic Redundancy Check

Length Byte

Das Length Byte (TL) gibt die Anzahl der im ATS übertragenen Bytes an. Hierbei wird TL selbst mitgezählt, nicht jedoch die beiden CRC-Byte. Die Länge des ATS darf die im Request for Answer to Select (RATS) angegebene maximale Rahmenlänge (FSD) nicht überschreiten. Das bedeutet, dass der Maximalwert von TL nicht größer sein darf als FSD-2.

Format Byte

Ist die im TL angegebene Länge größer als 1, wird als Nächstes das Format Byte T0 gesendet. T0 besteht aus drei Teilen:

- Das höchstwertige Bit b8 hat den Wert 0. Der Wert 1 ist für zukünftige Verwendung reserviert.
- Die Bit b7 bis b5 zeigen die Anwesenheit von nachfolgenden Interface Bytes TC1, TC2, und TC3 an.
- Das niederwertige Halbbyte b4 bis b1 heißt FSCI (*Frame Size for proximity Card Integer*). Es codiert die maximale Rahmengröße FSC (*Frame Size for proximity Card*), dievon der Karte empfangen werden kann. Der Default-Wert für FSCI ist 2 und bedeutet eine maximale Rahmengröße von 32 Bytes.

Tabelle 3.32 Die Codierung des Format Byte T0.

b8	b7	b6	b5	b4	b3	b2	b1	Bedeutung
0	...	...	...	...	...	...	...	Ist auf 0 gesetzt, 1 ist RFU.
0	1	X	X	...	...	...	...	TC1 wird übertragen.
0	X	1	X	...	...	...	...	TB1 wird übertragen.
0	X	X	1	...	...	...	...	TA1 wird übertragen.
0	...	...	...	X	X	X	X	Maximale Rahmengröße FSCI

Tabelle 3.33 Codierung von b4 bis b1 des Parameters FSCI

FSCI	0	1	2	3	4	5	6	7	8	9-F
FSC in Byte	16	24	32	40	48	64	96	128	256	RFU>256

Interface Byte TA1

Das Interface Byte TA1 besteht aus vier Teilen:

- Das höchstwertige Bit zeigt an, ob unterschiedliche Divisoren D für die beiden Übertragungsrichtungen möglich sind. Aus dem Divisor D ergibt sich die etu (elementary time unit, das ist die Zeitdauer für 1 Bit) nach der Formel: 1 etu = $128/(D \cdot f_C)$. Der Anfangswert für den Divisor D ist 1, womit sich eine etu von 128/fC ergibt.
- Die Bit b7 bis b5 werden DS (Divisor Send) genannt und geben die unterstützten Bitraten der Karte für die Übertragungsrichtung Karte zu Terminal an.
- Bit b4 ist auf 0 gesetzt. Der Wert 1 ist RFU.
- Die Bit b3 bis b1 werden DR (Divisor Receive) genannt und geben die unterstützten Bitraten der Karte für die Übertragungsrichtung Terminal zu Karte an.

Die Auswahl eines Divisors wird vom Terminal im nachfolgenden Kommando PPS Request vorgenommen.

Tabelle 3.34 Die Codierung des Interface Byte TA1.

b8	b7	b6	b5	b4	b3	b2	b1	Bedeutung
0	...	...	...	0	...	...	...	Es werden unterschiedliche D für die beiden Übertragungsrichtungen unterstützt.
1	...	...	...	0	...	...	...	Es wird nur ein und dasselbe D für die beiden Übertragungsrichtungen unterstützt.
...	1	...	...	0	...	...	...	DS = 8 wird unterstützt.
...	...	1	...	0	...	...	...	DS = 4 wird unterstützt.
...	...	...	1	0	...	...	...	DS = 2 wird unterstützt.
...	...	...	...	0	1	...	...	DR = 8 wird unterstützt.
...	...	...	...	0	...	1	...	DR = 4 wird unterstützt.
...	...	...	...	0	...	...	1	DR = 2 wird unterstützt.

Interface Byte TB1

Im Interface Byte TB1 werden Parameter zur Definition der Rahmenwartezeit und der Start-up Rahmenschutzzeit übertragen. Folglich besteht das Interface Byte TB1 aus zwei Teilen:

- Das höherwertige Halbbyte b8 bis b5 heißt FWI (*Frame Waiting time Integer*) und bestimmt die Rahmenwartezeit FWT (*Frame Waiting Time*). Die Bedeutung und Berechnung der FWT ist im folgenden Kapitel beschrieben.
- Das niederwertige Halbbyte b4 bis b1 heißt SFGI (*Start-up Frame Guard time Integer*) und dient zur Berechnung der Start-up-Rahmenschutzzeit SFGT (*Start-up Frame Guard Time*). Die SFGT ist die Zeit, welche die Karte nach Aussendung des ATS benötigt, um wieder empfangsbereit für den nächsten Rahmen zu sein. Der Wert 15 von SFGT ist RFU. Der Wert 0 bedeutet, dass die Karte keine besondere SFGT benötigt. Für die Werte 1 bis 14 berechnet sich die SFGT nach der Formel:

$$SFGT = (256 \times 16/f_C) \times 2^{SFGI}.$$

Der Default-Wert für SFGI ist 0. Der Maximalwert $SFGT_{MAX}$ ergibt sich zu ungefähr 4949 ms.

b8	b7	b6	b5	b4	b3	b2	b1
FWI				SFGI			

Bild 3.126 Codierung des Interface Byte TB1.

Interface Byte TC1

Das Interface Byte TC1 zeigt spezielle Optionen des Protokolls an. Das TC1 besteht aus folgenden Teilen:

- Die höchstwertigen Bit b8 bis b3 sind auf 0 gesetzt. Alle anderen Werte sind RFU.
- Die Bits b2 und b1 zeigen an, welche optionalen Felder im Prolog-Feld der Karte unterstützt werden (siehe folgendes Kapitel). Das Terminal ist nicht gezwungen, alle Felder, die unterstützt werden, auch zu übertragen, es kann also Felder weglassen. Felder, die nicht von der Karte unterstützt werden, dürfen aber auf keinen Fall vom Terminal an die Karte übertragen werden. Der Default-Wert für Bit b2 ist 1, und für Bit b1 ist er 0. Das bedeutet, dass CID unterstützt und NAD nicht unterstützt wird.

b8	b7	b6	b5	b4	b3	b2	b1
0	0	0	0	0	0	b2 = 1 bedeutet: CID (*Card Identifier*) wird unterstützt	b1 = 1 bedeutet: NAD (*Node Address*) wird unterstützt

Bild 3.127 Codierung des Interface Byte TC1.

Historical Bytes

Die Historical Bytes T1 bis Tk sind optional. Ihr Inhalt ist in ISO/IEC 7816-4 definiert (siehe auch Abschnitt 6.2 Answer to Reset – ATR). Aus der maximalen Länge des ATS lässt sich die maximal mögliche Anzahl der Historical Bytes ermitteln.

Protokoll und Parameter-Selektion

Wenn die Karte im ATS anzeigt, dass sie veränderbare Parameter unterstützt, kann das Terminal mithilfe des Kommandos PPS (*Protocol and Parameter Selection*) die Parameter für das nachfolgende Protokoll verändern. Für den Fall, dass die Karte keine veränderbaren Parameter unterstützt, muss sie auch das PPS-Kommando nicht unterstützen (siehe auch Bild 3.122). Das Protokoll wird dann unmittelbar mit unveränderten Parametern fortgesetzt.

Nehmen wir an, dass die Karte die Unterstützung veränderbarer Parameter im ATS angezeigt hat. Das Terminal kann nun auswerten, ob es selbst die von der Karte signalisierten Veränderungsmöglichkeiten nutzen will. Ist dies der Fall, sendet es ein PPS request mit folgendem Aufbau:

1. Byte	2. Byte	3. Byte	4. Byte	5. Byte
PPSS Start Byte	PPS0 Parameter 0 zeigt die Anwesenheit von PPS1 an.	PPS1 Parameter 1 codiert DRI und DSI.	CRC1	CRC2

Bild 3.128 Aufbau von PPS Request (Protocol und Parameter Selection Request).

Start Byte

Das Start Byte PPSS besteht aus zwei Teilen:

- Das höherwertige Halbbyte b8 bis b5 ist auf 'D' gesetzt und identifiziert das PPS. Alle anderen Werte sind RFU.
- Das niederwertige Halbbyte b4 bis b1 heißt CID (*Card Identifier*) und definiert die logische Nummer der adressierten Karte.

Parameter 0

Der Parameter 0 (PPS0) zeigt an, ob das optionale Parameterbyte 1 vorhanden ist.

Tabelle 3.35 Codierung des Parameter 0. Alle anderen Werte für b8 bis b6 und b4 bis b1 sind RFU.

b8	b7	b6	b5	b4	b3	b2	b1
0	0	0	Wenn b5 = 1 ist, wird PPS1 übertragen.	0	0	0	1

Parameter 1

Im Parameter 1 wird angezeigt, welche der Divisoren DS und DR, die im Interface Byte TA1 angeboten wurden, ausgewählt sind. Hierdurch wird die Übertragungsgeschwindigkeit für die folgende Datenübertragung festgelegt.

Tabelle 3.36 Die Codierung von Parameter 1 (PPS1)

b8	b7	b6	b5	b4	b3	b2	b1	Bedeutung
0	0	0	0	DSI		DRI		DRI und DSI codieren den Divisor D.

Tabelle 3.37 Die Codierung von D durch DRI und DRS

DRI / DSI	0	0	0	1	1	0	1	1
D	1		2		4		8	

Protocol and parameter selection response
Die Karte bestätigt den korrekten Empfang des protocol and parameter selection request durch ein protocol and parameter selection response, welches nur aus dem Startbyte PPSS und den beiden Checkbyte CRC1 und CRC2 besteht. Von nun an benutzen das Terminal und die Karte die ausgewählten Parameter für die Datenübertragung.

Activation frame waiting time
Um unnötig lange Wartezeiten bei Übertragungsstörungen zu vermeiden, ist in der ISO/IEC 14443-4 die maximale Zeit definiert, die verstreichen darf, bevor eine Typ-A-Karte eine Antwort sendet, nachdem sie das Ende eines Rahmens empfangen hat. Diese Zeit wird activation frame waiting time genannt und ist auf 65 536/f_C (~4 833 µs) festgelegt. Antwortet eine Karte nicht innerhalb dieser Zeit, weiß das Terminal, dass die Kommunikation mit der Karte gestört ist.

Fehlererkennung und Fehlerkorrektur
In einem System mit kontaktlosen Karten ist damit zu rechnen, dass häufiger Störungen in der Datenübertragung auftreten, als man es bei Karten mit Kontakten gewohnt ist. So ist es zum Beispiel bei kontaktbehafteten Karten relativ selten, dass eine Karte während der Kommunikation mit dem Terminal diesem entnommen wird. Bei kontaktlosen Karten können Unterbrechungen während der Kommunikation häufiger vorkommen, da ja die Karte während des Betriebs im Aktionsfeld bewegt und dabei auch unabsichtlich die Reichweite des Terminal verlassen werden kann. Es ist also wichtig, dass Methoden verfügbar sind, solche Unterbrechungen schnell zu erkennen und die Kommunikation möglichst wieder definiert aufnehmen zu können.

Damit man nicht bei jeder Störung die Kommunikation ganz abbrechen und von vorn beginnen muss, sind in der ISO/IEC 14443-4 für die Phase der Protokollaktivierung von Typ-A-Karten Regeln für die Fehlererkennung und Fehlerkorrektur definiert. Auf die Erläuterung dieser Regeln wird hier verzichtet und auf das entsprechende Kapitel der Norm verwiesen, wo diese übersichtlich dargestellt sind.

3.6.3.4.2 Halbduplexblockprotokoll nach ISO/IEC 14 443-4

Das im Teil 4 von ISO/IEC 14 443-4 definierte Übertragungsprotokoll berücksichtigt die speziellen Anforderungen der Anwendung von kontaktlosen Karten. So bietet es zum Beispiel dem Terminal die Möglichkeit, eine Kommunikation mit mehreren Karten parallel durchzuführen.

Das Protokoll unterstützt ähnlich wie das Übertragungsprotokoll T = 1 für kontaktbehaftete Karten eine saubere Schichtentrennung nach dem OSI-Referenzmodell. Schichtentrennung bedeutet, dass Daten höherer Schichten völlig transparent von niedrigeren Schichten übertragen werden. Das Protokoll baut auf den Rahmen auf, die im Teil 3 der Norm für die beiden Kartentypen (Typ A und Typ B) definiert sind.

Es werden 4 Schichten unterschieden:

- Die physikalische Schicht (Byteübertragung nach 14 443-3),
- die Leitungsschicht (*Data link layer*, Austausch von Datenblöcken nach 14 443-4),
- die Sitzungsschicht (*Session layer*), die mit der Leitungsschicht verbunden ist, um minimalen Overhead zu erzeugen, und
- die Anwendungsschicht, in der Kommandos abgearbeitet werden.

Das Blockprotokoll startet, nachdem die Aktivierungssequenz abgeschlossen wurde. Das erste Senderecht hat das Terminal. Dies bedeutet, dass die Karte nach der Aktivierungssequenz warten muss, bis sie einen Block vom Terminal empfängt. Die Karte reagiert auf empfangene Blöcke innerhalb der definierten Rahmenwartezeit mit dem Senden eines Response-Blocks. Danach liegt das Senderecht wieder beim Terminal und die Karte wechselt wieder in den Empfangsmode. Die Kommunikation wird dann mit abwechselndem Senderecht entsprechend weitergeführt.

Das Protokoll erlaubt die gleichzeitige Aktivierung mehrerer Karten durch ein Terminal. Es kann zwischen mehreren Karten hin- und herschalten, ohne jedes Mal Zeit für die Deaktivierung einer Karte und die Aktivierung einer anderen Karte zu verbrauchen.

Wie bereits gesagt, ist in Systemen mit kontaktlosen Karten die Wahrscheinlichkeit einer Störung der Datenübertragung höher, als bei vergleichbaren Systemen mit kontaktbehafteten Karten. Es ist deshalb besonders wichtig, dass die Kommunikation zwischen Karten und Terminal in möglichst kurzer Zeit erfolgt. Die Datenübertragungsrate ist mit 106 kBit/s (Default-Wert) relativ hoch, verglichen mit 9,6 kBit/s (Default-Wert) bei den Übertragungsprotokollen T = 0 und T = 1.

Blockaufbau

Die Übertragungsblöcke setzen sich aus einem führenden Prologfeld, dem Informationsfeld (optional) und einem nachgestellten Epilogfeld zusammen. Das optionale Informationsfeld enthält die Daten für die Anwendungsschicht.

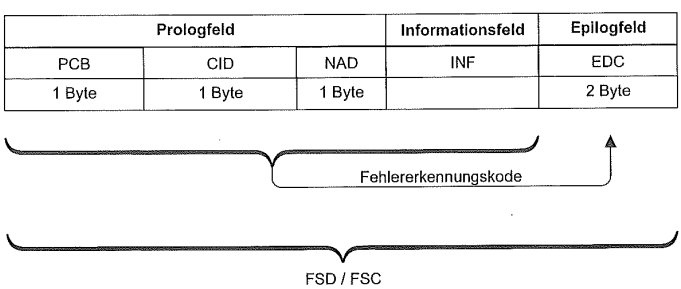


Bild 3.129 Aufbau eines Blockes.

Es werden 3 Blocktypen unterschieden:

- Die Informationsblöcke, die zum transparenten Austausch von Daten der Anwendungsschicht benutzt werden;
- Die Empfangsbestätigungsblöcke (R-Block), die kein Informationsfeld enthalten und die zur positiven oder negativen Empfangsbestätigung dienen;

- Die Systemblöcke (S-Block), in denen Steuerinformationen für das Protokoll ausgetauscht werden. Es sind 2 Typen von S-Blocks definiert: ein S-Block zur Verlängerung der Rahmenwartezeit mit einem Informationsfeld von einem Byte sowie ein DESELECT-Block ohne Informationsfeld, mit dem die Karte in den HALT-Zustand versetzt werden kann.

Prologfeld

Das Prologfeld besteht aus dem Protokollkontrollbyte (PCB), dem optionalen Card Identifier (CID) und der optionalen Knotenadresse (NAD).

Die Codierung des Protokollkontrollbytes ist in Tabelle 3.38 beschrieben.

Tabelle 3.38 Codierung des Protokollkontrollbytes PCB.

Bit	I-Block PCB	R-Block PCB	S-Block PCB	
			DESELECT	WTX
b8	0	1	1	
b7	0	0	1	
b6	0 (1 ist RFU)	1	0	1
b5	1 signalisiert Chaining	0 = ACK, 1 = NAK	0	1
b4	1 signalisiert CID folgt	1 signalisiert CID folgt	1 signalisiert CID folgt	
b3	1 signalisiert NAD folgt	0 (keine NAD)	0 (keine NAD)	
b2	1	1 (0 ist RFU)	1 (0 ist RFU)	
b1	Block Nummer	Block Nummer	0 (1 ist RFU)	

Card Identifier Field – CID

Das Card-identifier-Feld dient dazu, eine spezifische Karte zu identifizieren, und enthält außerdem Informationen zur Leistungsversorgung der Karte. Die höchstwertigen Bits b8 und b7 zeigen den Leistungspegel an, mit dem die Karte vom Terminal versorgt wird. Die Bits b6 und b5 sind auf 0 gesetzt und für zukünftige Verwendung reserviert. Die Bits b4 bis b1 codieren die Kartenidentifikationsnummer.

Tabelle 3.39 Codierung des Card-identifier CID

b8	b7	b6	b5	b4	b3	b2	b1
Leistungsanzeige		0, 1 = RFU	0, 1 = RFU	CID			

Die Codierung des CID ist für Typ-A-Karten im PPS-Kommando und für Typ-B-Karten im ATTRIB-Kommando definiert. Bei der Auswertung des CID gelten für die Karte folgende Regeln:

- Eine Karte, die den CID nicht unterstützt, ignoriert alle Blöcke, die einen CID enthalten.
- Eine Karte, die CID unterstützt, antwortet auf Blöcke, die ihre CID enthalten, und sendet dabei ihre eigene CID zurück. Blöcke mit anderen CID ignoriert die Karte und für den Fall, dass die Karte den CID = 0 hat, antwortet die Karte auf Blöcke ohne CID mit Blöcken ohne CID.

Diese Regeln ermöglichen es dem Terminal, gleichzeitig mit mehreren aktiven Karten zu kommunizieren, ohne die nicht angesprochenen Karten deaktivieren zu müssen.

Mit den Bit b8 und b7 wird die Anzeige der Leistung folgendermaßen codiert:

Tabelle 3.40 Codierung der Leistungsanzeige (*Power level indication*)

b8	b7	Bedeutung
0	0	Die Karte unterstützt die Leistungsanzeige nicht
0	1	Die Leistung reicht für die volle Funktionalität nicht aus
1	0	Ausreichende Leistung für volle Funktionalität
1	1	Mehr als ausreichende Leistung für volle Funktionalität

Knotenadresse – NAD

Das dritte Byte des Prologfeldes wird Knotenadresse genannt und dient dazu, verschiedene logische Verbindungen zwischen Karte und Terminal aufzubauen und zu adressieren. Der Gebrauch der Knotenadressen entspricht dem bei kontaktbehafteten Karten und ist in der ISO/IEC 7816-3 definiert und in Kapitel 6 beschrieben.

Informationsfeld INF

In den I-Blocks dient das Informationsfeld als Container für die Daten der Anwendungsschicht. Der Inhalt dieses Feldes wird vollständig transparent übertragen.

In den S-Blocks dient das Informationsfeld zur Steuerung der Verlängerung der Rahmenwartezeit.

Rahmenwartezeit FWT (*Frame Waiting Time*)

Um einen definierten Abbruch der Kommunikation mit einer nicht antwortenden Karte in möglichst kurzer Zeit zu erreichen, ist eine Rahmenwartezeit FWT definiert, die der Blockwartezeit BWT beim T = 1 Protokoll für kontaktbehaftete Karten entspricht. Die Rahmenwartezeit ist die maximale Zeitdauer zwischen dem Ende eines Rahmens, der von dem Terminal gesendet wurde, und dem Start des Rahmens, mit dem die Karte antwortet. Läuft diese Zeit ab, ohne dass die Karte eine Antwort sendet, kann das Terminal von einer Fehlfunktion der Karte ausgehen, und es erhält das Senderecht zurück, um Mechanismen zur Fehlererkennung einzuleiten Wie bereits beschrieben, enthält der ATS für Typ-A-Karten im Interface-Byte TB1 den Wert FWI (frame waiting time integer), mit dem sich die Rahmenwartezeit folgendermaßen berechnet:

$$FWT = (256 \times 16 / f_C) \times 2^{FWI}.$$

Für Typ-B-Karten ist FWI im answer to request ATQB definiert. Für FWI = 0 ergibt sich der Minimalwert von FWT zu ungefähr 302 µs. Der Maximalwert FWT_{MAX} ergibt sich mit FWI = 14 zu ungefähr 4949 ms. Für die Auswahl der Rahmenwartezeit muss man die übliche Ausführungsdauer von Kommandos durch die Karte berücksichtigen. Bemisst man die Wartezeit zu großzügig, verzögert man die Detektion einer Karte, die nicht mehr antwortet. In der Praxis kommt es vor, dass der Großteil der Kommandos relativ schnell durch die Karte

beantwortet werden kann, einzelne Kommandos, die zum Beispiel die Berechnung von Kryptoalgorithmen beinhalten, jedoch deutlich mehr Zeit beanspruchen. Um in solchen Fällen die Rahmenwartezeit nicht generell auf einen hohen Wert legen zu müssen, gibt es den Mechanismus der Wartezeitverlängerung, mit dem die Karte für die Beantwortung eines einzelnen Kommandos eine Verlängerung der Rahmenwartezeit anfordern kann.

Verlängerung der Rahmenwartezeit

Zur Anforderung einer Verlängerung der FWT sendet die Karte einen speziellen S-Block S(WTX) request und erhält darauf vom Terminal einen passenden S-Block S(WTX) response als Bestätigung dieser Anfrage. Eine Ablehnung dieser Anfrage durch das Terminal ist nicht erlaubt.

Die Dauer der Verlängerung der Rahmenwartezeit wird dem Terminal durch das Byte im Informationsfeld des S-Blocks S(WTX) request mitgeteilt. Dieser Wert, multipliziert mit der Rahmenwartezeit, ergibt die neue temporäre Rahmenwartezeit, die nur für die Bearbeitung des aktuellen Kommandos gilt.

Tabelle 3.41 Codierung des INF-Feldes in einem S(WTX) request

b8	b7	b6	b5	b4	b3	b2	b1
Leistungsanzeige		WTXM (1 bis 59, 0 und 60 bis 63 sind RFU)					

Tabelle 3.42 Codierung des INF-Feldes in einem S(WTX) response

b8	b7	b6	b5	b4	b3	b2	b1
0	0	WTXM					

Die Messung der temporären Rahmenwartezeit beginnt mit dem Ende des vom Terminal ausgesandten S(WTX) response. Die temporäre Rahmenwartezeit FWT_{TEMP} berechnet sich nach der Formel:

$$FWT_{TEMP} = FWT \times WTXM$$

Ergibt sich aus der Formel ein Wert größer als FWT_{MAX} (~4 949ms), gilt anstelle der berechneten FWT_{TEMP} der Wert FWT_{MAX}.

Blockverkettung (Chaining)

Die Funktion der Blockverkettung ermöglicht es, jeweils einem der beiden Kommunikationspartner Datenblöcke zu übertragen, die nicht in einen einzelnen Rahmen passen, indem die Daten auf mehrere I-Blöcke aufgeteilt und nacheinander übertragen werden. Jeder einzelne dieser I-Blöcke hat eine Länge, die kleiner oder gleich der durch FSC bzw. FSD festgelegten Rahmenlänge ist.

Im Protokollkontrollbyte PCB des ersten Blocks der Kette wird das Chaining Bit gesetzt. Dieses zeigt dem Empfänger an, dass die Blockverkettungsfunktion verwendet wird und dass der nachfolgende Block verkettete Daten enthält.

Wenn der Empfänger diesen ersten Informationsblock der Kette korrekt empfangen hat, signalisiert er mit einem R-Block, dessen Blocknummer der gerade empfangenen Block-

nummer entspricht, dass der Block korrekt empfangen wurde und er bereit ist, den nächsten Block der Kette zu empfangen. Daraufhin kann der nächste Block gesendet werden. Diese Hin und Her von I- und R-Blöcken setzt sich so lange fort, bis der Sender einen I-Block überträgt, bei dem das Chaining Bit nicht gesetzt ist. Nach Empfang dieses Blocks weiß der Empfänger, dass nun alle Daten der Anwendungsschicht angekommen sind, und er kann diesen Block verarbeiten und die zugehörige Antwort senden.

Deaktivierung einer Karte
Wenn die Datenübertragung zwischen Terminal und Karte vollständig abgeschlossen ist, versetzt das Terminal die Karte in den HALT-Zustand. Hierzu dient das DESELECT-Kommando, welches in einem S-Block übertragen wird. Die Karte antwortet hierauf mit S(DESELECT)-Response-Block und fällt in den HALT-Zustand.

Fehlerbehandlung
Das Blockprotokoll verfügt über ähnliche Fehlererkennungsmechanismen wie das T = 1 Protokoll, die bei Übertragungsfehlern eine Resynchronisation in mehreren Stufen ermöglichen. Die genauen Regeln für den Protokollablauf sind dem Teil 4 der ISO/IEC 14443 zu entnehmen. Im Anhang der Norm sind auch ausführliche Beispiele für fehlerfreie Protokollabläufe sowie für die Fehlerbehandlung zu finden.

3.6.4 Vicinity Integrated Circuits Cards nach ISO/IEC 15 693

Die ISO/IEC-Norm 15 693 mit dem genauen Titel „Identification cards – Contactless integrated circuit(s) cards - Vicinity cards" beschreibt Eigenschaften und Funktionsweise von kontaktlosen Chipkarten mit einer Reichweite bis zu 1 Meter. Dieser Kartentyp wird zum Beispiel bevorzugt für die Zugangskontrolle eingesetzt, da die Funktionsreichweite von ca. einem Meter einen Betrieb ermöglicht, bei dem die Karte nicht in die Hand genommen werden muss, sondern zum Beispiel in der Handtasche bleiben kann. Bisher hat sich diese Norm in Chipkartensystemen kaum durchgesetzt, weshalb auf eine ausführliche Beschreibung an dieser Stelle verzichtet wird.

3.6.5 Prüfmethoden für kontaktlose Chipkarten

In der Norm ISO/IEC 10 373 sind alle Testverfahren für ID-Karten mit und ohne Chip zusammengefasst. Wie man der folgenden Auflistung entnehmen kann, enthalten drei Teile dieser Norm spezielle Testmethoden für kontaktlose Karten. Die ISO/IEC 10 373 besteht aus 6 Teilen:

ISO/IEC 10 373 Identification cards – Test methods

- 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: Testverfahren für Close-coupling-Chipkarten

In diesem Teil der Norm sind Verfahren zum Testen der physikalischen Schnittstelle von kontaktlosen Close-coupling-Chipkarten nach ISO/IEC 10536 beschrieben. Es sind Testmittel in Form von Referenzspulen und kapazitiven Koppelflächen definiert, mit denen die Energie- und Datenübertragung zwischen der Chipkarte und dem Terminal gemessen und auf Konformität zu den in der Norm (ISO/IEC 10536) festgelegten Werten überprüft werden kann.

3.6.5.2 Part 6: Testverfahren für Proximity-coupling-Chipkarten

Im Teil 6 der Norm sind Verfahren zum Testen der physikalischen Schnittstelle von Proximity-coupling-Chipkarten nach ISO/IEC 14443 beschrieben. Die hierzu erforderlichen speziellen Testmittel: eine Kalibrationsspule, ein Prüfaufbau zur Messung der Lastmodulation sowie eine Referenzkarte sind in der Norm definiert. Für folgende Eigenschaften der Karte bzw. des Terminals sind Prüfverfahren in der Norm beschrieben:

- Widerstandsfähigkeit der Karte gegen elektrostatische Entladung;
- Amplitude der Lastmodulation sowie die Funktionalität der Karte innerhalb ihres definierten Modulationsbereiches, wie im Basisstandard beschrieben;
- die vom Terminal generierte Feldstärke;
- der Modulationsindex sowie das Ein- und Ausschwingverhalten des vom Terminal erzeugten Signals

Es sei darauf hingewiesen, dass die genaue und reproduzierbare Messung des Lastmodulationssignals wegen der schwachen Signalpegel schwierig ist, und es bleibt zu hoffen, dass bald geeignete Messgeräte im Markt angeboten werden. Bis dahin ist es ratsam, die Unterstützung der Lieferanten von Karten bzw. Terminals anzufordern und frühzeitig entsprechende Lieferbedingungen zu vereinbaren.

3.6.5.3 Part 7: Testverfahren für Vicinity-coupling-Chipkarten

Im Teil 7 der Norm sind Verfahren zum Testen der physikalischen Schnittstelle von Vicinity-cards nach ISO/IEC 15693 beschrieben. Die Prüfmittel und Testverfahren entsprechen weitgehend denen in Teil 6 der Norm. Es ergibt sich lediglich ein anderer Aufbau der Referenzkarte wegen der unterschiedlichen Frequenz des Hilfsträgers.

4 Informationstechnische Grundlagen

„Chipkarten sind kleine Computer in Scheckkartenformat und ohne Mensch-Maschine-Schnittstelle." Diese Aussage trifft den Kern der Sache. Die spezifischen Eigenschaften von Chipkarten gegenüber allen anderen Kartentypen sind durch den in den Kartenkörper implantierten Mikrocontroller bestimmt.

Der Kartenkörper aus Kunststoff ist in seiner Funktion vor allem ein Träger für den Mikrocontroller. Natürlich können zusätzlich zum Mikrocontroller weitere Kartenelemente integriert sein, doch ist dies für die eigentlichen Chipkarten-Funktionen nicht notwendig. Um nun die Eigenschaften dieses kleinen Computers und der darauf aufbauenden informationstechnischen Mechanismen verstehen zu können, sind einige Grundkenntnisse der Informatik notwendig.

Es soll hier nicht Expertenwissen vermittelt werden. Was auch nicht notwendig ist, um die Grundzüge der Verfahren und Techniken im Zusammenhang mit Chipkarten zu begreifen. Das Grundlagenwissen, das hier in diesem Kapitel dargeboten ist, reicht zum Verstehen völlig aus. Die beschriebenen spezifischen Sachverhalte gehen deshalb nur so weit in die Tiefe, wie es für das Verständnis der Zusammenhänge notwendig ist.

Beinahe die Hälfte dieses Kapitels ist kryptografischen Verfahren gewidmet, die im Umfeld von Chipkarten eingesetzt werden. Wissen über Kryptologie war noch vor wenigen Jahren mit dem Schleier des Geheimnisvollen und Unbekannten umgeben. Dies änderte sich jedoch zunehmend, und es existiert mittlerweile auch umfangreiche Literatur zu diesem Thema. Analog zu den Teilen des Kapitels über allgemeine Informationstechnik sind hier nur Grundlagen beschrieben, die für das Verständnis von kryptografischen Algorithmen und Protokollen bei Chipkarten notwendig sind. Für detaillierte Informationen sei auf die bekannten Werke, wie beispielsweise von Bruce Schneier [Schneier 96] und Alfred Menezes [Menezes 97], verwiesen. Eine weitere ergiebige Quelle an aktueller Information über Kryptologie ist das World Wide Web mit den Homepages von Forschungseinrichtungen (z. B.: [GMD, Semper]), Normungsinstanzen (z. B.: [ETSI, IEC, ISO]), Behörden (z. B.: [BSI, NSA]), Firmen (z. B.: [Ascom, Certicom, Counterpane, R3, RSA]), Vereinen (z. B.: [CCC, Teletrust] oder an dem Thema interessierten Menschen (z. B.: [Gutmann, Tatu]).

4.1 Strukturierung von Daten

Die Speicherung oder Übertragung von Daten erfordert zwangsläufig immer eine exakte Definition der verwendeten Daten und ihrer Struktur. Nur so ist es möglich, Datenelemente wieder zu erkennen und auch zu interpretieren. Datenstrukturen mit fester Länge und nicht änderbarer Reihenfolge bringen regelmäßig Systeme zum „Platzen". Das beste Beispiel dafür ist die Umstellung vieler europäischer Währungen in Euro. Alle Systeme und Datenstrukturen, bei denen die Währungskennzeichnung unveränderlich festgelegt war, mussten mit großem Aufwand erweitert werden. Die gleiche Problematik herrscht bei vielen Chipkarten-Anwendungen. Feste Datenstrukturen, die erweitert oder gekürzt werden sollen, verursachen über kurz oder lang immer erheblichen Aufwand.

Das Problem, Daten zu strukturieren, besteht allerdings schon seit Jahren, und es gibt genügend Lösungsansätze dazu. Einer, der in der Chipkartenwelt sehr populär ist und auch allgemein in der Informationstechnik immer mehr Anwendung findet, stammt aus der Datenübertragung. Es ist dies die abstrakte Syntax-Notation 1 (*abstract syntax notation one*),

die kurz als ASN.1 bezeichnet wird. ASN.1 ist eine codierungsunabhängige Beschreibung von Datenobjekten, die ursprünglich aus dem Bereich des Transfers von Daten zwischen verschiedenen Rechnersystemen stammt. Eine Alternative zu ASN.1 wäre die Strukturierung der Daten mit XML (*Extensible Markup Language*), doch hat dies bei realen Anwendungen im Chipkartenumfeld bisher noch nicht Fuß gefasst.

Im Prinzip ist ASN.1 eine Art von künstlicher Sprache, die nicht zur Beschreibung von Programmen geeignet ist, sondern zur Beschreibung von Daten und ihrer Struktur. Genormt ist die Syntax in der ISO/IEC 8824, die Codierungsregeln befinden sich in der ISO/IEC 8825. Beide Normen sind eine Weiterentwicklung der Empfehlung X.409, die von der CCITT definiert wurde.

ASN.1 im Detail zu beschreiben würde ein ganzes Buch erfordern, weshalb hier überblickshaft nur auf einige wesentliche Aspekte eingegangen und des Weiteren auf einschlägige Literatur wie beispielsweise Walter Gora [Gora 98] verwiesen wird.

ASN.1 besitzt eine Reihe von elementaren (*primitiv*) Datentypen und zusammengesetzten (*constructed*) Datentypen. Weiterhin besteht die Möglichkeit, über Makros die Syntax von ASN.1 zu erweitern, um damit beinahe beliebige Ergänzungen von ASN.1 vorzunehmen. Die Tabellen 4.2, 4.3 und 4.4 zeigen ein einfaches Beispiel für die Verwendung von ASN.1 inklusive Definition und Kodierung der Daten.

Das Grundkonzept der Kodierung von Daten mit ASN.1 besteht darin, dass Datenobjekte mit einer vorangestellten eindeutigen Kennzeichnung und einer Längenangabe versehen

Tabelle 4.1 Einige oft benutzte Datentypen von ASN.1

Datentyp	Art	Bedeutung
BOOLEAN	elementar	Wahrheitswert Ja/Nein
INTEGER	elementar	negative und positive Ganzzahl
OCTED STRING	elementar	Sequenz von Bytes (1 Byte = 1 Oktett zu 8 Bit)
BIT STRING	elementar	Sequenz von Bits
SEQUENCE	zusammengesetzt	Zusammenfassung mehrerer Komponenten zu einem neuen Datentyp

Tabelle 4.2 Ein einfaches Beispiel für eine Datendefinition mittels ASN.1

`SC_Controller ::= SEQUENCE {`	Definition eines neuen Datentyps für SC_Controller.
`  Name IA5String,`	Der Name des Mikrocontrollers ist ein ASCII-String.
`  CPUType CPUPower,`	Bei CPUType wird auf die Definition von CPUPower verwiesen.
`  NPU BOOLEAN,`	Wahrheitswert als Ja/Nein-Aussage, ob ein Koprozessor (NPU) vorhanden ist.
`  EEPROMSize INTEGER,`	Die Größe des EEPROMs ist ein ganzzahliger Integerwert.
`  RAMSize INTEGER,`	Die Größe des RAMs ist ein ganzzahliger Integerwert.
`  ROMSize INTEGER}`	Die Größe des ROMs ist ein ganzzahliger Integerwert.
`CPUPower ::= ENUMERATED {`	Definition eines neuen Datentyps als Aufzähltyp für CPUPower.
`  8Bit (8),`	Der mögliche Auswahlwert für den 8 Bit CPU Typ.
`  16Bit (16),`	Der mögliche Auswahlwert für den 16 Bit CPU Typ.
`  32Bit (32)}`	Der mögliche Auswahlwert für den 32 Bit CPU Typ.

Tabelle 4.3 Die Datendefinition aus Tabelle 4.2 mit Daten eines spezifischen Mikrocontrollers gefüllt

`SuperXS SC_Controller ::= {`	Spezifische Ausprägung des Datentyps SC-Controller mit den Daten für SuperXS.
`  Name "XS 8 Bit",`	Der Name des Mikrocontrollers ist „XS 8 Bit".
`  CPUType 8,`	Es handelt sich um eine 8 Bit CPU.
`  NPU true,`	Ein Koprozessor (NPU) ist vorhanden.
`  EEPROMSize 1024,`	Die Größe des EEPROMs beträgt 1 024 Byte.
`  RAMSize 256,`	Die Größe des RAMs beträgt 256 Byte.
`  ROMSize 8192}`	Die Größe des ROMs beträgt 8 192 Byte.

Tabelle 4.4 Die Daten eines spezifischen Mikrocontrollers aus Tabelle 4.3 gemäß ASN.1 mit den BER kodiert

`'30 1C'`	Kennzeichen '30' für eine Sequenz mit der Länge 28 Byte (= '1C').
`  '16 08 58 53 20 38 20 42 69 74'`	Kennzeichen '16' für einen IA5String mit der Länge 8 Byte (= '08') und dem Inhalt '58 53 20 38 20 42 69 74' (= "XS 8 Bit").
`  '0A 01 08'`	Kennzeichen '0A' für einen Aufzähl-Datentyp mit der Länge 1 Byte (= '01') und dem Inhalt '08'.
`  '01 01 FF'`	Kennzeichen '01' für einen Boolean Datentyp mit der Länge 1 Byte (= '01') und dem Inhalt 'FF', was dem Wert true entspricht.
`  '02 02 04 00'`	Kennzeichen '02' für einen Integer Datentyp mit der Länge 2 Byte (= '02') und dem Inhalt '04 00' (= 1 024).
`  '02 02 01 00'`	Kennzeichen '02' für einen Integer Datentyp mit der Länge 2 Byte (= '02') und dem Inhalt '01 00' (= 256).
`  '02 02 20 00'`	Kennzeichen '02' für einen Integer Datentyp mit der Länge 2 Byte (= '02') und dem Inhalt '02 00' (= 8 192).

werden. Die recht aufwendige Syntax der Beschreibungssprache erlaubt es auch, eigene Datentypen zu definieren sowie Datenobjekte zu verschachteln. Die ursprüngliche Idee, eine allgemein gültige Syntax zu schaffen, auf deren Basis Daten zwischen zwei grundverschiedenen Rechnersystemen ausgetauscht werden können, kommt im Chipkartenbereich wenig zum Einsatz. Hier findet momentan lediglich ein sehr kleiner Teil der gesamten zur Verfügung stehenden Syntax Verwendung. Dies ist vor allem durch den beschränkten Speicherplatz in den Chipkarten bedingt.

In der ISO/IEC 8825 sind die grundlegenden Codierungsregeln BER (*basic encoding rules*) für ASN.1 zusammengefasst. Diese daraus resultierenden Datenobjekte werden als BER-TLV-codierte Datenobjekte bezeichnet. BER-codierte Datenobjekte besitzen ein Kennzeichen, eine Längenangabe, den eigentlichen Datenteil und optional eine Endekennung. Nach diesen Codierungsregeln sind auch einige Bits im Kennzeichen vorbelegt. Dies ist in Bild 4.1 detailliert dargestellt. Eine Untermenge der grundlegenden Codierungsregeln BER sind die DER (*distinguished encoding rules*). Diese geben unter anderem an, wie die Längenangabe zu codieren ist (1, 2 oder 3 Byte lang). Ein grundlegender Überblick zu BER und DER findet sich bei Burton Kaliski [Kaliski 93].

Die Codierung von ASN.1-Objekten findet somit in der klassischen TLV-Struktur statt. Dabei bedeutet das „T" (*tag*) die Kennzeichnung des Datenobjektes, das „L" (*length*) die Länge und „V" (*value*) die eigentlichen Daten.

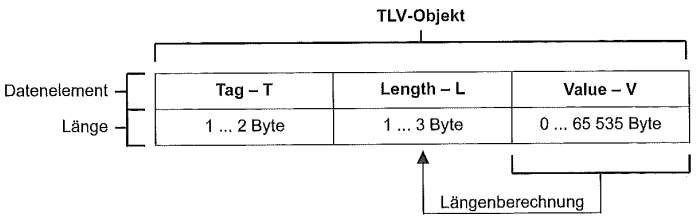


Bild 4.1 Das Prinzip der BER-basierten TLV-Codierung nach ASN.1.

Das erste Feld in einer TLV-Struktur ist das Kennzeichen des im V-Teil folgenden Datenobjektes. Damit nun nicht jeder Anwender seine eigenen Kennzeichen definieren muss und somit der Inkompatibilität Tür und Tor geöffnet ist, existieren Normen, die für verschiedene und auch öfter benötigte Datenstrukturen die entsprechenden „tags" festlegen. In der ISO/IEC 7816-6 sind beispielsweise die Kennzeichen von Datenobjekten für allgemeine Industrieanwendungen definiert. Für Secure Messaging sind sie hingegen in der ISO/IEC 7816-4 festgelegt, und bei EMV gibt es wiederum eigene. Es ist also keineswegs so, dass ein spezieller Tag überall für das Gleiche Datenelement benutzt wird, doch im Wesentlichen ist eine Vereinheitlichung im Gange.

In den beiden höherwertigen Bits des Kennzeichens wird die Klasse codiert, zu der das nachfolgende Datenobjekt gehört. Die Klasse gibt an, ob es sich um generelle Definitionen von Datenobjekten handelt (*universal class*), wie beispielsweise für Integerwerte oder Zeichenketten. Die „application class" zeigt an, ob das Datenobjekt zu einer bestimmten Anwendung oder Norm (z. B. ISO/IEC 7816-6) gehört. Die beiden weiteren Klassen, „context specific class" und „private class", gehören in den Bereich von nicht genormten Anwendungen.

Das den beiden Bits für die Klasse folgende Bit zeigt an, ob das gekennzeichnete Datenobjekt aus weiteren Datenobjekten zusammengesetzt ist. Die fünf niederwertigen Bits sind das eigentliche Kennzeichen. Da es aufgrund des eingeschränkten Adressraumes nur die Werte von 0 bis 30 annehmen kann, existiert die Möglichkeit, durch Setzen aller fünf Bits auf das nachfolgende Byte zu verweisen. In diesem sind dann alle Werte von 31 bis 127 zulässig. Das Bit 8 verweist auf zukünftige Anwendungen und darf demnach noch nicht gesetzt sein.

Die Anzahl der benötigten Längenbytes ist in Tabelle 4.1 dargestellt. In der Norm wird noch der Begriff Template (deutsch: Schablone) definiert. Ein Template ist ein Datenobjekt, das als Hülle oder Container für weitere Datenobjekte dient. Für den Bereich von branchenübergreifenden Anwendungen von Chipkarten sind die Kennzeichen für mögliche Datenobjekte in der ISO/IEC 7816-6 definiert. Der Bereich des Zahlungsverkehrs mit Chipkarten deckt die ISO 9992-2 ab.

Diese Methode der Codierung von Daten hat einige Eigenschaften, die sich gerade im Chipkartenbereich sehr vorteilhaft auswirken. Da der Speicherplatz im Regelfall immer zu klein ist, kann durch die Verwendung von Datenobjekten auf der Grundlage von ASN.1 erheblich Speicherplatz gespart werden. Denn durch die TLV-Codierung ist es möglich, Daten mit variabler Länge ohne große Komplikationen zu übertragen und abzuspeichern. Dies führt zu einer sehr ökonomischen Verwendung von Speicher.
Am Beispiel der TLV-Codierung eines Namens sei dies noch einmal verdeutlicht.

Spätere Erweiterungen von Datenstrukturen können bei ASN.1 sehr einfach vorgenommen werden, da in der vorhandenen Datenstruktur nur ein zusätzliches TLV-codiertes Da-

Tabelle 4.5 Die Codierung des Kennzeichens bei ASN.1

Byte 1	b8	b7	b6	b5	b4	b3	b2	b1	Bedeutung
	0	0	...	...	...	...	...	...	universal class
	0	1	...	...	...	...	...	...	application class
	1	0	...	...	...	...	...	...	context specific class
	1	1	...	...	...	...	...	...	private class
	...	...	0	...	...	...	...	...	einfaches Datenobjekt (*primitive*)
	...	...	1	...	...	...	...	...	zusammengesetztes Datenobjekt (*constructed*)
	...	....	...	X	X	X	X	X	Nummer des Kennzeichens (0 ... 30)
	...	...	...	1	1	1	1	1	Es existiert ein nachfolgendes Byte (Byte 2), das das Kennzeichen festlegt.
Byte 2	**b8**	**b7 ... b1**							**Bedeutung**
	0	31 ... 127							Nummer des Kennzeichens

Tabelle 4.6 Aufbau der DER-basierten Längenangabe bei ASN.1

Byte 1	Byte 2	Byte 3	Bedeutung
0 ... 127	---	---	für diesen Wertebereich wird 1 Byte als Längenangabe benutzt
'81'	128 ... 255	---	für diesen Wertebereich werden 2 Byte als Längenangabe benutzt
'82'	256 ... 65 535		für diesen Wertebereich werden 3 Byte als Längenangabe benutzt

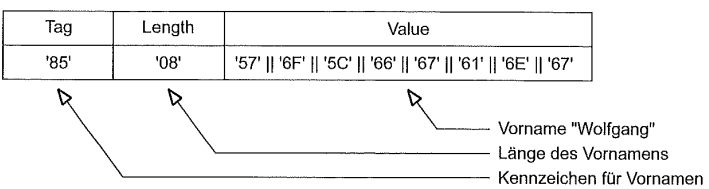


Bild 4.2 Codierung des Vornamens „Wolfgang" mittels einer TLV Struktur.

tenobjekt eingefügt werden muss. Die Kompatibilität zu den älteren Versionen bleibt dabei vollständig erhalten, solange die früheren TLV-Objekte nicht entfernt werden. Genauso verhält es sich mit neuen Versionen von Datenstrukturen, bei denen Änderungen gegenüber der vorherigen Codierung vorgenommen wurden. Durch die Änderung der Kennzeichnung, also des „tags", ist dies ohne weiteres möglich. Ebenso lassen sich gleiche Daten mit unterschiedlicher Codierung sehr einfach darstellen. Zusammengefasst ergeben alle diese Vorteile die Begründung, warum gerade im Chipkartenbereich die ASN.1-Syntax auf der Basis einer TLV-Codierung so beliebt ist.

Der Hauptnachteil von ASN.1-Datenobjekten ist der bei geringen Nutzdaten erhebliche Aufwand an Verwaltungsdaten. Sind die Nutzdaten beispielsweise nur 1 Byte lang, dann benötigt man zwei zusätzliche Bytes (*tag* und *length*), um dieses eine Datenbyte zu verwalten. Je länger jedoch die Nutzdaten sind, desto günstiger wird das Verhältnis. Die ASN.1-strukturierten Daten in der deutschen Krankenversichertenkarte sind ein gutes Beispiel da-

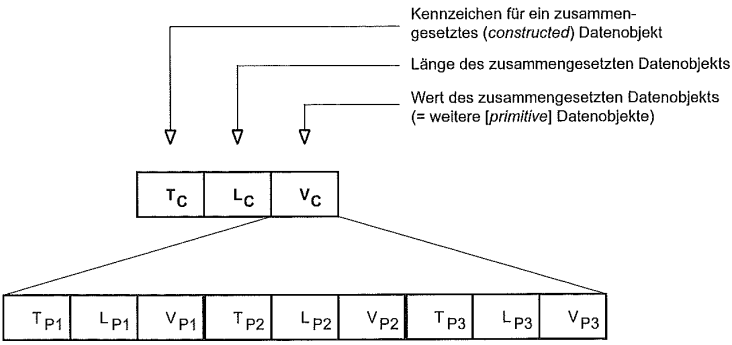


Bild 4.3 Prinzipielle Struktur beim Aufbau von zusammengesetzten (*constructed*) TLV-codierten Datenstrukturen aus mehreren einfachen (*primitive*) TLV-codierten Datenobjekten. Die Indizes „C" und „P" stehen für „constructed" und „primitive".

für. Die Nutzdaten betragen zwischen 70 und 212 Byte. Die dafür notwendigen Verwaltungsdaten haben eine Gesamtanzahl von 36 Byte. Dies ergibt einen Verwaltungsaufwand für die Nutzdaten zwischen 17 und 51 Prozent.

An einem Beispiel sei dies alles nochmals rekapituliert. Angenommen, man will in einer Datei mit transparenter Datenstruktur Familiennamen, Vornamen und Titel speichern. Unabhängig von einer korrekten ASN.1-Beschreibung werden die TLV-codierten Daten folgende Struktur haben (die im Beispiel verwendeten Kennzeichen sind frei gewählt und entsprechen damit auch keiner einschlägigen Norm):

Variante 1

T	L	V	T	L	V	T	L	V
'85'	'07'	"Manfred"	'87'	'05'	"Meier"	'84'	'04'	"Ing."

Variante 2

T	L	V	T	L	V	T	L	V
'84'	'04'	"Ing."	'85'	'07'	"Manfred"	'87'	'05'	"Meier"

Variante 3

T	L	V	T	L	V	T	L	V
'87'	'05'	"Meier"	'85'	'07'	"Manfred"	'84'	'04'	"Ing."

Bild 4.4 Beispiele für die Unabhängigkeit der Reihenfolge innerhalb einer TLV-Struktur.

Bei der Auswertung dieser Datenstruktur vergleicht der Rechner das erste Kennzeichen mit allen ihm bekannten. Stimmt es mit einem überein, erkennt er das erste Datenobjekt als Vorname. Die dazugehörige Länge kann er im folgenden Byte lesen. Die anschließenden Bytes sind dann das eigentliche Datenobjekt, d .h. der Vorname. An diesem schließt sich das nächste TLV-Objekt mit dem ersten Byte, dem Kennzeichen für den Familiennamen, an. Der Computer geht hier bei der Erkennung genauso vor wie beim ersten Datenobjekt.

Besteht nun die Notwendigkeit, die Datenstruktur zu erweitern, z. B. mit einem Namenszusatz, kann man einfach ein neues Element in die vorhandene Struktur einfügen. Die Position der Einfügung ist dabei nicht von Bedeutung. Die nun erweiterte Datenstruktur

bleibt aber dennoch voll kompatibel mit der älteren Version, da das neue Element ein eigenes Kennzeichen erhält und damit eindeutig gekennzeichnet ist. Programme, die nur die alten Kennzeichen erkennen, stören sich an dem neuen Kennzeichen nicht, da es für sie unbekannt ist und definiert übersprungen werden kann. Andere Programme, die auch das neue Tag erkennen, können es damit auch auswerten und haben aber auch keine Probleme mit der alten Struktur.

4.2 Codierung alphanumerischer Daten

Die in den Dateien und Datenobjekten von Chipkarten gespeicherten alphanumerischen Daten können in den unterschiedlichsten Formaten abgelegt sein. Zu einem Teil lässt sich dies auf intensive Speicherplatzoptimierungsmaßnahmen und mangelnde Abstimmung der einzelnen Anwendungen und Spezifikationen untereinander zurückführen, zum anderen Teil jedoch auf den Siegeszug der Chipkarten in Ländern außerhalb Westeuropas, welche eigene Schriftzeichen benutzen. Dabei mussten die ursprünglich verwendeten 7-Bit- und 8-Bit-Zeichensätze durch leistungsfähigere Codierungsschemata für alphanumerische Daten ersetzt werden.

4.2.1 7-Bit-Code

Mit einer 7-Bit-Codierung können insgesamt $2^7 = 128$ Zeichen dargestellt werden. Der weltweit am meisten benutzte 7-Bit-Code ist in ISO/IEC 646 spezifiziert und vor allem als ASCII Code (*American Standard Code for Information Interchange*) bekannt. Die Bedeutung von ASCII nimmt seit Jahren stetig ab, da die Anzahl der darstellbaren Schriftzeichen viel zu gering ist.

Im GSM-System wurde im Übrigen zur Darstellung von alphanumerischen Daten ein an ASCII angelehntes 7-Bit-Alphabet festgelegt. Es ist in der GSM-Spezifikation GSM 03.38 definiert und hat im Bereich der mobilen Telekommunikation große Bedeutung erlangt.

4.2.2 8-Bit-Code

Der am häufigsten verwendete 8-Bit-Code ($2^8 = 256$ Zeichen) ist nach ISO/IEC 8859 genormt und wurzelt im 7-Bit-ASCII Code. Er setzt sich aus zwei 7-Bit-Code-Tabellen zusammen, die beide sowohl Steuerzeichen als auch Schriftzeichen festlegen. Die niederwertigere der beiden Codetabellen ist dabei identisch mit ASCII und immer gleich. Die höher-wertige Codetabelle kann je nach ländertypischem Zeichensatz unterschiedlich sein. Die wohl bekannteste höherwertige Codetabelle ist Latein 1 und deckt die spezifischen Schriftzeichen der westeuropäischen Länder ab. Latein 2 hingegen codiert die besonderen Schriftzeichen der osteuropäischen Länder. Die ISO/IEC 8859 besteht insgesamt aus 16 Teilen, die noch eine Reihe weiterer höherwertiger Codetabellen für Schriftzeichen verschiedenster Sprachen enthalten.

Die Schriftzeichen von ISO/IEC 8859 mit Latein-1-Codetabelle finden sich in etwas geänderter Codierung auch unter DOS als Codetabelle 850 nach IBM-Register, als so genanntes PC-ASCII und unter Windows als ANSI-Code.

Der im Bereich von Großrechnern immer noch weit verbreite 8 Bit breite EBCDIC Code (*extended binary coded decimal interchange code*) findet im Übrigen bei Chipkarten keine Anwendung.

Tabelle 4.7 Die für das GSM-System festgelegte und an den ASCII-Zeichensatz angelehnte 7-Bit Standard-Zeichentabelle nach GSM 03.38. Die folgenden Abkürzungen wurden verwendet: Wagenrücklauf (*CR – carriage return*), Zeilenvorschub (*LF – line feed*) und Leerzeichen (*SP – space*). In den Kästchen ist oben mittig das jeweilige Zeichen dargestellt, links unten der 7-Bit-Zeichencode in dezimaler Notation und rechts unten in hexadezimaler Notation.

@ 0 '00'	Δ 16 '10'	SP 32 '20'	0 48 '30'	¡ 64 '40'	P 80 '50'	¿ 96 '60'	p 112 '70'
£ 1 '01'	_ 17 '11'	! 33 '21'	1 49 '31'	A 65 '41'	Q 81 '51'	a 97 '61'	q 113 '71'
$ 2 '02'	Φ 18 '12'	" 34 '22'	2 50 '32'	B 66 '42'	R 82 '52'	b 98 '62'	r 114 '72'
¥ 3 '03'	Γ 19 '13'	# 35 '23'	3 51 '33'	C 67 '43'	S 83 '53'	c 99 '63'	s 115 '73'
è 4 '04'	Λ 20 '14'	¤ 36 '24'	4 52 '34'	D 68 '44'	T 84 '54'	d 100 '64'	t 116 '74'
é 5 '05'	Ω 21 '15'	% 37 '25'	5 53 '35'	E 69 '45'	U 85 '55'	e 101 '65'	u 117 '75'
ù 6 '06'	Π 22 '16'	& 38 '26'	6 54 '36'	F 70 '46'	V 86 '56'	f 102 '66'	v 118 '76'
ì 7 '07'	Ψ 23 '17'	' 39 '27'	7 55 '37'	G 71 '47'	W 87 '57'	g 103 '67'	w 119 '77'
ò 8 '08'	Σ 24 '18'	(40 '28'	8 56 '38'	H 72 '48'	X 88 '58'	h 104 '68'	x 120 '78'
Ç 9 '09'	Θ 25 '19'	) 41 '29'	9 57 '39'	I 73 '49'	Y 89 '59'	i 105 '69'	y 121 '79'
LF 10 '0A'	Ξ 26 '1A'	* 42 '2A'	: 58 '3A'	J 74 '4A'	Z 90 '5A'	j 106 '6A'	z 122 '7A'
Ø 11 '0B'	1) 27 '1B'	+ 43 '2B'	; 59 '3B'	K 75 '4B'	Ä 91 '5B'	k 107 '6B'	ä 123 '7B'
ø 12 '0C'	Æ 28 '1C'	, 44 '2C'	< 60 '3C'	L 76 '4C'	Ö 92 '5C'	l 108 '6C'	ö 124 '7C'
CR 13 '0D'	æ 29 '1D'	- 45 '2D'	= 61 '3D'	M 77 '4D'	Ñ 93 '5D'	m 109 '6D'	ñ 125 '7D'
Å 14 '0E'	ß 30 '1E'	. 46 '2E'	> 62 '3E'	N 78 '4E'	Ü 94 '5E'	n 110 '6E'	ü 126 '7E'
å 15 '0F'	É 31 '1F'	/ 47 '2F'	? 63 '3F'	O 79 '4F'	§ 95 '5F'	o 111 '6F'	à 127 '7F'

4.2.3 16-Bit-Code (Unicode)

Codes mit 16 Bit Breite ermöglichen die Darstellung von $2^{16} = 65536$ Zeichen. Der einzige Vertreter ist dabei der von der privaten Initiative Unicode Konsortium [Unicode] als Industriestandard definierte Unicode.

Die ersten 256 Zeichen von Unicode sind identisch mit ISO/IEC 8859 Latein 1, sodass es hier zumindest in der Zeichencodierung eine Aufwärtskompatibilität gibt. Die Anzahl

der mit einem 16 Bit Code darstellbaren Zeichen reicht zwar aus, um die Schriftzeichen der wichtigsten lebenden Sprachen darzustellen, doch leider nicht für alle Schriftzeichen.

Um dies zu kompensieren, wurde in der aktuellen Unicode Version 3.0 eine Art Escape-Sequenz (*surrogate pairs*) in der 16 Bit Zeichenbreite eingeführt, mit der es nun über ein zusätzliches Byte möglich ist, bis zu einer Million Zeichen darzustellen.

4.2.4 32-Bit-Code (UCS)

Aufgrund der ursprünglichen Limitierung auf 65536 Zeichen bei Unicode wurde ein erweiterter Zeichencode notwendig, der diese im täglich Gebrauch allerdings unkritische, Begrenzung umging. ISO/IEC 10646 legt einen 32-Bit-Code namens UCS (*universal character set*) fest, mit dem sich $2^{32} = 4\ 294\ 967\ 296$ Zeichen darstellen lassen, wobei jedoch nur die Hälfte (= 2 147 483 648 Zeichen) davon benutzt werden.

Die vier Bytes von UCS haben in sinkender Wertigkeit sortiert die Namen Gruppe (*group*), Ebene (*plane*), Reihe (*row*) und Zelle (*cell*). UCS setzt sich demzufolge aus 128 Gruppen zu 256 Ebenen zu 256 Reihen zu 256 Zellen zusammen. Eine Ebene legt damit 65536 Zeichen fest. Die unterste Ebene, d. h. Gruppe 0 und Ebene 0, wird BMP (*basic multilingual plane*) genannt und ist identisch mit dem Unicode. Die unterste Reihe, d. h. Gruppe 0, Ebene 0 und Reihe 0, entspricht infolgedessen auch automatisch dem Zeichensatz von ISO/IEC 8859 Latein 1, und die ersten 128 Zeichen sind auf diese Weise auch identisch der ASCII-Codierung.

Ein kurzes Beispiel soll dies verdeutlichen: Der Buchstabe "A" in 7 Bit ASCII und 8 Bit ISO/IEC 8859 Latein 1 wird als '30' codiert. Da die ersten 256 Zeichen von Unicode identisch mit ISO/IEC 8859 Latein 1 sind, hat demnach "A" in Unicode dargestellt den Wert '00 30' und im 4 Byte breiten UCS Code den Wert '00 00 00 30'.UCS ist der einzige Zeichencode, mit dem es möglich ist, alle Zeichen aller lebenden und toten Sprachen mit einem bestimmten Zahlenwert eindeutig darzustellen. Deshalb ist UCS trotz seines Speicherplatzbedarfs von 4-Byte pro Zeichen das wichtigste zukünftige Codierungsschema.

Zur Übertragungskodierung von 32-Bit-UCS- und 16-Bit-Unicode-Zeichen gibt es drei übliche Schemata, welche als UTF (*UCS transformation format*) bezeichnet werden. Mit UTF-8 werden die Zeichen in eine Sequenz von Byte variabler Länge übersetzt, deren 7 niederwertigste Bits ASCII entsprechen. UTF-16 verwendet 16 Bits zu Kodierung und entspricht damit der BMP von UCS und dem 16-Bit-Unicode. Ein anderer Ausdruck für UTF-16 ist auch UCS-2, da dabei 2 Byte zur Kodierung benutzt werden. UTF-32 entspricht der üblichen 4 Byte-Darstellung von UCS und wird deshalb auch UCS-4 genannt.

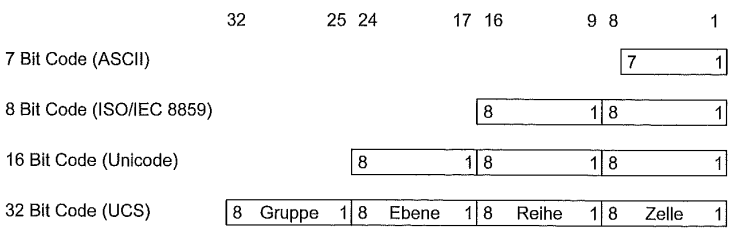


Bild 4.5 Die Zusammenhänge zwischen den weltweit am häufigsten benutzten 7, 8, 16 und 32 Bit Codes für alphanumerische Zeichen.

4.3 SDL-Symbolik

In diesem Buch wird zur Beschreibung von Zuständen und Zustandsübergängen die SDL-Notation benutzt. Diese ist seit einigen Jahren im Chipkartenbereich eine immer häufiger eingesetzte Symbolik zur Beschreibung von zustandsorientierten Mechanismen, beispielsweise von Kommunikationsprotokollen. SDL ist die Abkürzung von Specification and Description Language und in der CCITT-Empfehlung Z.100 detailliert beschrieben.

Die SDL-Symbolik ähnelt dem der üblichen Flussdiagramme. Sie beschreibt jedoch keine Programmabläufe, sondern Zustände und Zustandsübergänge. Die SDL-Diagramme sind aus einzelnen genormten Symbolen zusammengesetzt, die mit Linien untereinander verbunden werden. Der Ablauf in den Diagrammen ist immer von links oben nach rechts unten, sodass die Verbindungslinien der einzelnen Symbole keine Kennzeichnung (Pfeilspitze) von Anfang und Ende benötigen.[1]

Vereinfacht kann man sich die Notation so vorstellen, dass SDL ein System beschreibt, das aus einer bestimmten Anzahl von Prozessen besteht. Jeder Prozess wiederum ist ein Zustandsautomat. Befindet sich der Automat in einem stabilen Zustand, so kann er ein Signal von außen empfangen. Abhängig von den empfangenen Daten wird ein bestimmter neuer Zustand erreicht. Dazwischen können zusätzliche Aktionen, wie Empfangen und Senden von Daten oder Berechnung eines Wertes, liegen.

Die folgenden zehn Symbole werden in diesem Buch benutzt. Sie stellen nur eine Auswahl aus der wesentlich umfangreicheren Symbolmenge von Z.100 dar, doch reichen sie als Grundlage für den Einsatz im Chipkartenbereich.

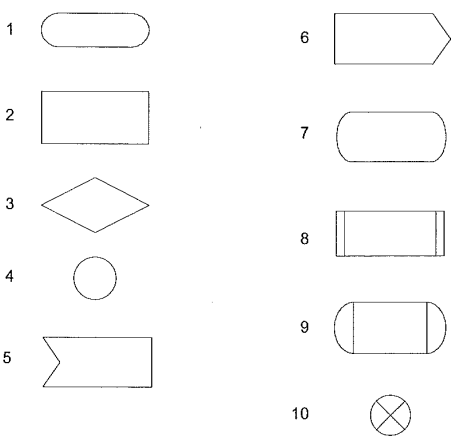


Bild 4.6 Die in diesem Buch verwendeten Symbole der SDL-Notation nach CCITT Z.100:

1 – Start	5 – Eingabe	8 – Unterprogramm
2 – Aufgabe	6 – Ausgabe	9 – Start Unterprogramm
3 – Entscheidung	7 – Zustand	10 – Ende Unterprogramm
4 – Marke		

[1] Ein ausführliches Beispiel für ein SDL-Diagramm findet sich in Abschnitt 6.4.2 „Übertragungsprotokoll T = 0".

Das Start-Zeichen (Nr. 1) symbolisiert den Anfang eines Prozesses. Mit ihm beginnt in den meisten Fällen ein SDL-Diagramm. Das Zeichen für eine Aufgabe (Nr. 2) erläutert durch den enthaltenen Text eine bestimmte Aktion näher. Es existiert bei diesem Symbol keine weitere detaillierte Beschreibung in Form eines Unterprogramms. Das nächste Zeichen, die Entscheidung (Nr. 3), erlaubt bei einem Zustandsübergang eine Abfrage, die entweder „JA" oder „NEIN" sein kann. Die Marke (Nr. 4) gibt eine Verbindung zu einem weiteren SDL-Diagramm an und wird hauptsächlich zur Aufteilung von großen Diagrammen auf mehrere kleinere benutzt.

Die beiden anschließenden Zeichen Eingabe (Nr. 5) und Ausgabe (Nr. 6) zeigen Schnittstellen nach außen auf. Im Inneren des Symbols werden die genauen Ein/Ausgabeparameter beschrieben. Zur Beschreibung eines Zustands dient das Symbol Zustand (Nr. 7). In ihm ist der jeweils erreichte Zustand angegeben.

Die nächsten drei Symbole beschreiben Unterprogramme. Die Nummer 8 – Unterprogramm – zeigt auf, dass der Inhalt dieses Kästchens an anderer Stelle noch genauer beschrieben ist. Die beiden Symbole Start (Nr. 9) und Ende (Nr. 10) eines Unterprogramms bilden den Rahmen für ausführlichere Darstellungen.

4.4 Zustandsautomaten

Unter einem Automaten stellt man sich gemeinhin eine Maschine vor, in die man eine Münze einwirft und dann einen Knopf drückt. Daraufhin kann man ein Fach öffnen und ihm etwas entnehmen. Etwas abstrakter ausgedrückt, definiert dieser Automat einen Ablauf mit verschiedenen Übergängen von Zuständen. Im Ausgangszustand wartet der Automat auf den Geldeinwurf, jede andere Aktion wie Drücken der Taste ruft keinerlei Aktion hervor. Erst der Einwurf der Münze überführt den Automaten vom Grundzustand in den Zustand „Geld eingeworfen". Der nächste Zustandsübergang kommt durch das Drücken der Taste zustande, woraufhin der Automat ein Fach freigibt.

In der Informatik kann man unter Verwendung von Graphen oder Petri-Netzen auf eine sehr anschauliche Art Zustandsautomaten beschreiben. Neben der Möglichkeit der Modellierung von Zustandsautomaten können die dadurch beschriebenen Systeme auf bestimmte Eigenschaften hin untersucht werden. Ziele dabei sind, etwaige mögliche Verklemmungen (*dead locks*) im Ablauf herauszufinden und die korrekten Befehlsabläufe sicherzustellen.

4.4.1 Grundlagen zur Automatentheorie

Es soll hier ein Überblick und eine Einführung über Graphen zur Zustandsbeschreibung von Chipkarten-Anwendungen gegeben werden.

Ein Graph stellt eine Menge von Zuständen und Beziehungen dieser Zustände zueinander dar. Die Zustände werden als Knoten dargestellt. Die Beziehungen der Zustände zueinander als Kanten. Weisen die Kanten eine Richtung, sprich einen Pfeil am Ende auf, so spricht man von gerichteten Kanten und damit von einem gerichteten Graphen. Der Pfeil gibt an, in welcher Richtung ein Zustandsübergang stattfinden kann. Die Platzierung von Knoten und Kanten in der grafischen Darstellung spielt keine Rolle bei der Interpretation des Graphen. Eine Folge von Knoten, die mit Kanten verbunden sind, bezeichnet man des

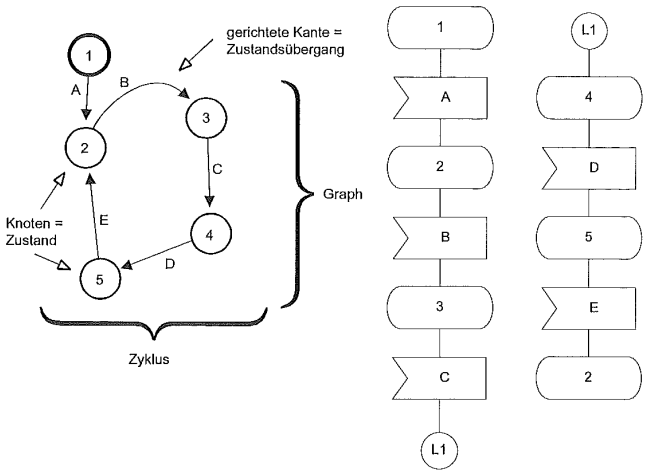


Bild 4.7 Beispiele für verschiedene Darstellungsarten eines Zustandsgraphen. Links ist ein gerichteter Zustandsgraph abgebildet, rechts ein äquivalentes SDL-Diagramm.

Tabelle 4.8 Darstellung des obigen Zustandsgraphen in Tabellenform

	von Zustand				
nach Zustand	**1**	**2**	**3**	**4**	**5**
1	---	---	---	---	---
2	A	---	---	---	E
3	---	B	---	---	---
4	---	---	C	---	---
5	---	---	---	D	---

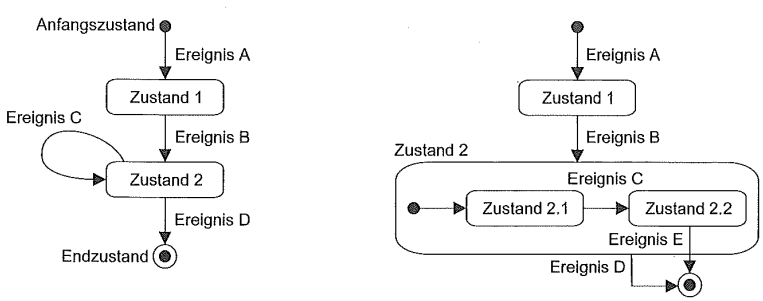


Bild 4.8 Beispiel für zwei Zustandsdiagramme nach UML. Das linke Bild zeigt einen einfachen sequenziellen Zustandsablauf und das rechte Bild ein Zustandsdiagramm mit Unterzuständen.

Weiteren als Pfad. Sind dabei der erste und letzte Knoten identisch und mehr als ein Knoten vorhanden, so nennt man diesen Pfad Zyklus.

Dies ist nur ein sehr kleiner Teil der Graphentheorie, doch reicht er im Wesentlichen aus, um die Zustände und die dazugehörigen Automaten in Chipkarten-Anwendungen zu beschreiben.

4.4.2 Praktische Anwendung

Gegenüber einfachen Speicherkarten besteht ein weiterer Vorteil von Mikroprozessor-Chipkarten darin, dass die Reihenfolge von Kommandos vorgegeben werden kann. Es besteht also die Möglichkeit, alle Kommandos in ihren Parametern und ihrer Reihenfolge genau festzulegen. Dies ist auch ein zusätzlicher Zugriffsschutz parallel zu den objektorientierten Zugriffsrechten auf Dateien. Allerdings sind die Möglichkeiten, die Chipkarten in diesem Bereich bieten, sehr unterschiedlich. Einfache Betriebssysteme können in der Regel keine Zustandsautomaten verwalten, wogegen in modernen Betriebssystemen anwendungsspezifische Zustandsautomaten mit Einbeziehung von Kommandoparametern definiert werden können.

Ein typisches Beispiel für einen einfachen Zustandsautomaten sind die beiden Kommandos, die für eine Authentisierung eines Terminals notwendig sind. Das erste Kommando fordert von der Karte eine Zufallszahl an. Damit aktiviert es den Zustandsautomaten, und dieser lässt als einziges nächstes Kommando ein Authentisierungskommando zu. Erhält die Karte das Kommando, so ist die Sequenz beendet, und jedes andere Kommando ist wieder erlaubt. Ist dies jedoch nicht der Fall, erhält die Karte also ein anderes als das erwartete Authentisierungskommando, dann erzeugt der Mikrozustandsautomat eine Fehlermeldung, und die Sequenz wird abgebrochen. Man muss dann die Kommandosequenz wieder von vorne beginnen.

Diese Arten von Zustandsautomaten haben in der Chipkarte einige große Vorteile. Weil nur sehr wenige Kommandos in einer fest definierten Sequenz vorgegeben werden, benötigen sie wenig Speicher und Programmaufwand. Für viele Anwendungsfälle reicht es aus, die Dateiinhalte mit den objektorientierten Zugriffsmechanismen zu schützen und sonst alle Kommandos frei in ihrer Reihenfolge zuzulassen. Lediglich einige Abläufe, wie beispielsweise die Authentisierung müssen in ihrer Reihenfolge vorgeschrieben werden. Dies kann sehr speicherökonomisch durch einen einfachen Zustandsautomaten geschehen.

Mit der Erweiterung der vorangehend beschriebenen einfachen Zustandsautomaten kann man alle Kommandos mit allen Parametern vor der Ausführung innerhalb eines definierten Graphen überprüfen. Je nach Ausführung des Automaten könnte man dann unter Umständen sogar auf den objektorientierten Zugriffsschutz an den Dateien verzichten, weil der Automat alle notwendigen Prüfungen vor der eigentlichen Kommandoausführung übernehmen könnte. Ein Fehler in der Definition des Zustandsgraphen hätte allerdings fatale Auswirkungen auf die Sicherheit des Systems. Weil die vollständige Fehlerfreiheit der Zustandsdefinitionen in komplexen Automaten nur mit hohem Aufwand nachweisbar ist, verzichtet man in der Praxis meist nicht auf den zusätzlichen Zugriffsschutz an den Dateien. Die korrekte Beschreibung aller Abläufe und aller Kommandos zu einer Chipkarte ist aufwendig und muss oft teilweise empirisch ermittelt werden.

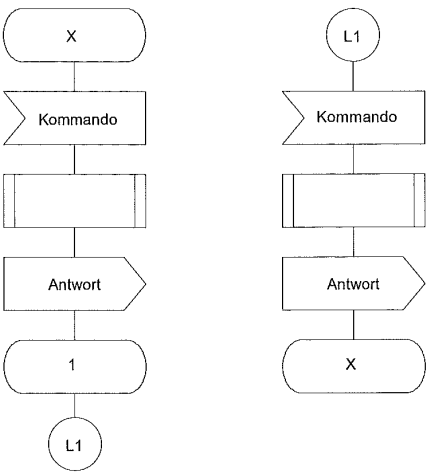


Bild 4.9 Ein Beispiel für einen einfachen Zustandsautomaten einer Chipkarte mit den beiden Zuständen X und 1.

Nach der Beschreibung der Vorteile von Zustandsautomaten müssen konsequenterweise auch die Nachteile genannt werden. Die Implementierung eines Zustandsautomaten mit der geforderten Mächtigkeit ist sowohl vom Entwurf als auch von der späteren Programmierung her gesehen sehr aufwendig. Der Bedarf an Programmspeicher alleine für den Automaten, der durch eine Speicherdarstellung eines Graphen gesteuert wird, ist beachtlich. Zusätzlich zu diesem Programmcode muss auch der jeweilige Graph im Speicher abgelegt werden. Die Größe dafür ist natürlich davon abhängig, wie komplex der auszuführende Graph ist. Die Informationsmenge, die in einem Graphen mit mehreren Zuständen und entsprechenden Zustandsübergängen enthalten ist, kann für Chipkarten-Verhältnisse groß werden. Zustandsautomaten für Chipkarten werden in der ISO/IEC 7816-9 behandelt. Dort werden so genannte ACDs (*access control descriptor*) beschrieben, in denen die in einem bestimmten Zustand erlaubten Kommandos mit ihren Parametern festgelegt werden. Das Chipkarten-Betriebssystem überwacht dann anhand dieser ACDs den festgeschriebenen Zustandsautomaten.

Um überblickshalber die Möglichkeiten eines Makroautomaten aufzuzeigen, ist im nachfolgendem Bild 4.10 ein Graph für eine kleine Anwendung abgebildet. Die Funktion kann wie folgt beschrieben werden:

Nach dem Reset befindet sich die Chipkarte im Grundzustand, der die Nummer 1 hat. In diesem Zustand ist die Auswahl jeder Datei mit SELECT FILE im Dateibaum erlaubt, dadurch tritt keine Zustandsänderung ein. Alle anderen Kommandos bis auf die PIN-Prüfung (VERIFY) sind verboten und werden von der Chipkarte mit einer Fehlermeldung quittiert. Nach einer positiven Überprüfung der PIN erreicht der Automat den Zustand 2.

In diesem Zustand sind zwei Kommandos erlaubt. Der erste Pfad führt über die Auswahl einer Datei (SELECT FILE) zum Zustand 3, in dem dann die ausgewählte Datei gelesen werden darf. Der zweite Pfad, der vom Zustand 2 abzweigt, führt nach der Anforderung einer Zufallszahl von der Karte durch das Terminal (ASK RANDOM) zu dem Zustand 4.

Jedes Kommando, außer EXTERNAL AUTHENTICATE, führt zurück zum Grundzustand 1. Konnte die Authentisierung des Terminals erfolgreich ausgeführt werden, erreicht die Chipkarte den Zustand 5. In diesem Zustand ist im Graph definiert, dass Dateien ausgewählt und geschrieben werden dürfen (SELECT FILE, UPDATE BINARY).

Die beiden Zustände 3 und 5 können in dieser Graphendefinition innerhalb der Sitzung nicht mehr verlassen werden, sie stellen die beiden Endzustände dar. Ein Übergang in den Zustand 1 ist nur durch einen Reset der Chipkarte möglich. Dies ist aber im Graphen nicht eingezeichnet, da alle Zustandsautomaten nur ein „Bewusstsein" während der aktuellen Sitzung haben. Es wird innerhalb des Zustandsautomaten keinerlei Information von einer Sitzung zu einer folgenden weitergegeben.

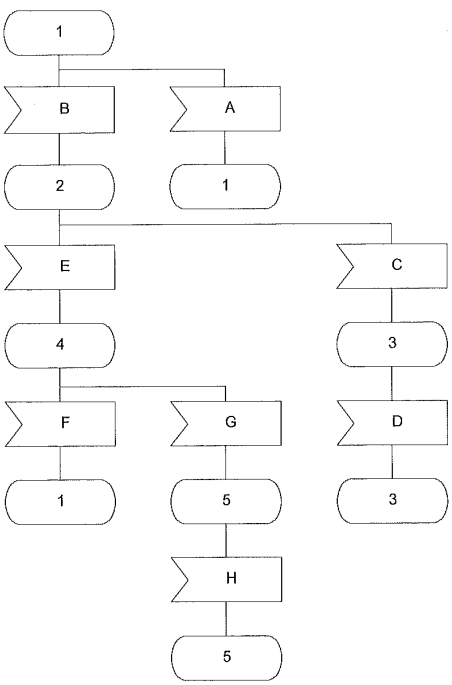


Bild 4.10 Beispiel für den aufwendigen Zustandsautomaten einer Chipkarte mit den folgenden Zuständen und Zustandsübergängen:

1 – Grundzustand
2, 4 – Übergangszustände
3, 5 – Endzustände
A – SELECT FILE
B – VERIFY
C – SELECT FILE
D – READ BINARY
E – ASK RANDOM
F – alle Kommandos außer G
G – EXTERNAL AUTHENTICATE
H – SELECT FILE/UPDATE BINARY

4.5 Fehlererkennungs- und Fehlerkorrekturcodes

Bei der Übertragung oder der Speicherung von Daten sollte eine Möglichkeit vorhanden sein, Veränderungen an diesen Daten zu erkennen. Gerade Programme müssen gegenüber Veränderungen abgesichert sein, da ein einziges verändertes Bit im Programmcode diesen zerstört bzw. den Programmablauf so verändert, dass die geforderte Funktionalität nicht mehr vorhanden ist. Vor allem der EEPROM-Speicher von Chipkarten ist gegenüber äußeren Einflüssen wie Hitze, Spannungsschwankungen oder Ähnlichem relativ empfindlich. Deshalb müssen gerade hier die sicherheitsrelevanten Teile geschützt sein, sodass unerwünschte Änderungen für das Betriebssystem erkennbar sind und in ihren negativen Auswirkungen abgefangen werden können.

Sehr empfindliche Dateninhalte – wie Programmcode, Schlüssel, Zugriffsbedingungen, Zeigerstrukturen und Ähnliches – müssen gegenüber Veränderungen gesichert sein. Dazu benutzt man Fehlererkennungscodes, die auch allgemein als EDC (*error detection code*) bezeichnet werden. Mit ihnen lassen sich Veränderungen in dem gesicherten Bereich mit einer je nach Code unterschiedlichen Wahrscheinlichkeit erkennen. Die erweiterte Form dazu sind die Fehlerkorrekturcodes, auch ECC (*error correction code*) genannt. Mit ihnen ist es möglich, nicht nur Fehler in den zu prüfenden Daten zu erkennen, sondern diese Fehler auch in begrenztem Maße wieder zu korrigieren.

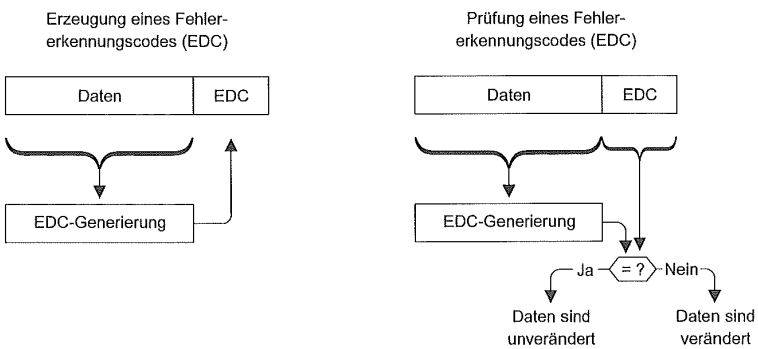


Bild 4.11 Prinzipieller Ablauf beim Einsatz eines fehlererkennenden Codes (*error detection code – EDC*).

Das Prinzip all dieser Codes ist es, den zu schützenden Daten eine Prüfsumme beizugeben. Üblicherweise befindet sich diese im Anschluss an die zu überwachenden Daten. Sie berechnet sich mit einem allgemein bekannten, also nicht geheimen Algorithmus. Mit dem durch die Berechnung erhaltenen EDC ist dann bei Bedarf eine Prüfung der Daten auf Veränderung möglich. Dies geschieht durch Vergleich der gespeicherten Prüfsumme mit einer neu ermittelten.

Gerade bei Fehlererkennung und -korrektur existieren die unterschiedlichsten Berechnungsverfahren. So sind etwa bei manchen die höherwertigen Bits besser geschützt, um dadurch die Auswirkungen im Wertebereich von Zahlen so weit wie möglich zu reduzieren. Der Einsatz dieser Algorithmen übersteigt jedoch in den meisten Fällen den Aufwand an Programmcode bei weitem. Es kommen deshalb meist nur Verfahren zur Anwendung,

bei denen bei der Erkennung von Fehlern keine Unterschiede zwischen höher- und niederwertigen Teilen eines Bytes gemacht werden und die byteorientiert arbeiten.

Die Fehlererkennungs- und -korrekturcodes sind den Message Authentication Codes (MAC) bzw. Cryptographic Checksums (CCS) sehr ähnlich. Es gibt jedoch einen grundlegenden Unterschied. Die EDC- oder ECC-Prüfsummen können von jedem berechnet und überprüft werden. Zur Berechnung der MAC- oder CCS-Prüfsummen benötigt man hingegen einen geheimen Schlüssel, da diese gegen Manipulation der Daten schützen sollen und nicht wie die EDC- oder ECC-Prüfsummen gegen unbeabsichtigte Verfälschung.

Der wohl bekannteste Fehlererkennungscode ist die Verwendung eines Paritätsbits, das bei vielen Übertragungsverfahren und auch bei manchen Speicherbausteinen an jedes zu schützende Byte angehängt wird. Vor der Berechnung der Parität muss festgelegt werden, ob mit gerader oder ungerader Parität gearbeitet wird. Bei der geraden Parität wird das Paritätsbit so gewählt, dass die Gesamtzahl der Einsen im Datenbyte und im Paritätsbit eine gerade Zahl ergibt. Bei ungerader Parität muss die Zahl der Einsen in Datenbyte und Paritätsbit ungerade sein.

Mit diesem Mechanismus der Paritätsberechnung ist die sichere Erkennung von einem falschen Bit pro Byte möglich. Die Korrektur eines Fehlers hingegen ist nicht möglich, da das Paritätsbit keine Aussage über die Position des veränderten Bits erlaubt.

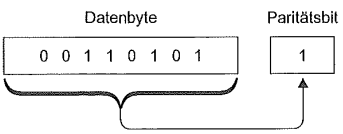


Bild 4.12 Beispiel für die Fehlererkennung unter Verwendung eines zusätzlichen Paritätsbits bei ungerader Parität.

Wären zwei Bits gleichzeitig pro Byte falsch, würde die Parität dadurch nicht verändert und der Fehler somit nicht erkannt. Ein weiterer Nachteil der Fehlererkennung durch Paritätsbildung ist der relativ große Overhead von einem Paritätsbit für acht Datenbits. Dies entspricht einem zusätzlichen Speicherbedarf von 12,5 %. Zudem ist es sehr schwierig, in byteweise organisierten Speichern mit zusätzlichen Paritätsbits zu arbeiten, weil es einen erheblichen Programmaufwand bedeuten würde. Deshalb wird die Fehlererkennung in Chipkartenspeichern nicht mit Paritätsbits durchgeführt. Besser geeignet sind dafür zum Beispiel XOR- oder CRC-Prüfsummen.

4.5.1 XOR-Prüfsummen

Die auch wegen der Berechnungsart als Längssummenprüfung (*longitudinal redundancy check – LRC*) bezeichnete XOR-Prüfsumme lässt sich einfach und auch schnell ermitteln. Beides sind wichtige Kriterien für einen in Chipkarten verwendeten Fehlererkennungscode. Zudem ist der Berechnungsalgorithmus äußerst simpel zu implementieren. Neben der Absicherung von Speicherinhalten finden XOR-Prüfsummen typischerweise auch bei der Datenübertragung (ATR, Übertragungsprotokoll T = 1) Anwendung.

Die Berechnung einer XOR-Prüfsumme erfolgt durch die aufeinander folgende logische XOR-Verknüpfung aller Datenbytes. Es wird also Datenbyte 1 mit Datenbyte 2 mit XOR

verknüpft. Das Ergebnis daraus wird dann wiederum mit Datenbyte 3 verknüpft und so weiter.

Stellt man die Prüfsumme direkt hinter die zu prüfenden Daten und errechnet dann nochmals eine Prüfsumme sowohl über die Daten als auch die vormals berechnete Prüfsumme, dann erhält man '00' als Ergebnis. Dies ist die einfachste Art zu überprüfen, ob Daten und Prüfsumme noch ihre ursprünglichen Werte haben und somit unverändert sind.

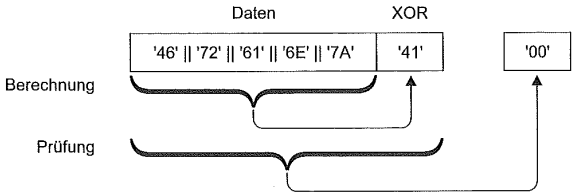


Bild 4.13 Die Bildung und Überprüfung einer XOR-Prüfsumme.

Die großen Vorteile von XOR-Prüfsummen liegen in der sehr schnellen Berechnung und in der Einfachheit des Algorithmus. Dieser Algorithmus ist so simpel aufgebaut, dass der Programmcode in Assembler dafür nur zwischen 10 und 20 Byte beträgt. Dies rührt auch daher, dass die logische XOR-Operation direkt als Maschinenbefehl in allen Prozessoren vorhanden ist. Zudem muss aufgrund verschiedener ISO-Normen (Datenübertragung mit T = 1) in fast jedem Chipkarten-Betriebssystem ein Algorithmus zur XOR-Berechnung implementiert sein, sodass er ohne zusätzlichen Aufwand auch noch für andere Zwecke verwendet werden kann.

Leider haben XOR-Prüfsummen auch einige gravierende Nachteile, und dies schränkt ihre praktische Anwendung erheblich ein. Sie sind vom Prinzip her nicht sehr sicher. Es ist damit z. B. nicht möglich, die Vertauschung von zwei Bytes innerhalb der Daten zu erkennen. Auch können sich Mehrfachfehler an der gleichen Bitposition in mehreren Bytes gegenseitig aufheben. Dies alles führt dazu, dass XOR-Prüfsummen ihr Haupteinsatzgebiet im Bereich der Datenübertragung haben und zur Überprüfung der Konsistenz von Speicherinhalten sehr verhalten eingesetzt werden. XOR-Prüfsummen lassen sich sehr schnell berechnen, da dazu in aller Regel eigene Maschinenbefehle vorhanden sind. Bei einem typischen 8-Bit-Chipkarten-Mikrocontroller mit 3,5 MHz Takt kann eine Geschwindigkeit von 1 µs/Byte erreicht werden, was einem Durchsatz von 1 MByte/s entspricht.

4.5.2 CRC-Prüfsummen

Ebenfalls aus dem Bereich der Datenkommunikation kommt das CRC-Verfahren (*cyclic redundancy check*), das aber gegenüber dem XOR-Verfahren wesentlich höherwertiger ist. Eine CRC-Prüfsumme ist aber gleichfalls nur ein Fehlererkennungscode, die Korrektur von Fehlern ist damit nicht möglich. Eingesetzt wird dieses Verfahren schon seit langer Zeit in Übertragungsprotokollen wie X/Z-Modem oder Kermit und vielfach als Hardwareimplementation in Festplattencontrollern. Es basiert auf der CCITT-Empfehlung V.41. Eine weitere Norm für CRC-Prüfsummen ist die ISO/IEC 13 239.

Die CRC-16-Prüfsumme wird durch ein rückgekoppeltes, zyklisches und 16 Bit langes Schieberegister erzeugt, bei CRC-32 ist das Schiebregister 32 Bit lang. Im Folgenden

wird nur auf den CRC-16 Bezug genommen, da dieses das übliche CRC-Prüfsummenverfahren bei Chipkarten ist. Die Rückkopplung im Schieberegister wird durch ein Generatorpolynom gesteuert. Mathematisch gesehen werden die zu prüfenden Daten als große Zahl aufgefasst und durch das Generatorpolynom dividiert. Der dabei erhaltene Divisionsrest stellt dann die Prüfsumme dar. Das CRC-16-Verfahren (d. h. CRC mit 16 Bit Länge) sollte nur bis zu einer Datengröße von 4 kByte eingesetzt werden, da darüber hinaus die Fähigkeit zur Fehlererkennung stark abfällt. Diese Einschränkung lässt sich aber durch eine Aufteilung der Daten in Blöcke bis zu maximal 4 kByte leicht umgehen. Alternativ ließe sich auch ein CRC-32-Verfahren (d. h. CRC mit 32 Bit Länge) einsetzen, das Einzelbitfehler in Daten bis 4 GByte erkennen kann.

Tabelle 4.9 Tabelle mit üblicherweise verwendeten Generatorpolynomen für die CRC-16 Berechnung.

Bezeichnung	Generatorpolynom
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	$G(x) = x^{12} + x^{11} + x^3 + x^2 + x + 1$

Es ist also bei CRC-Prüfsummen immer notwendig, das Generatorpolynom sowie auch noch zusätzlich die Vorbelegung des Schieberegisters zu kennen, da sonst die Berechnung der Prüfsumme nicht mehr nachvollzogen werden kann. Der Startwert des Schieberegisters ist in den überwiegenden Fällen null (z. B. ISO 3309), doch gibt es einige Übertragungsverfahren (z. B. CCITT Empfehlung X.25), die im Gegensatz dazu für alle Bits den Wert eins verwenden.

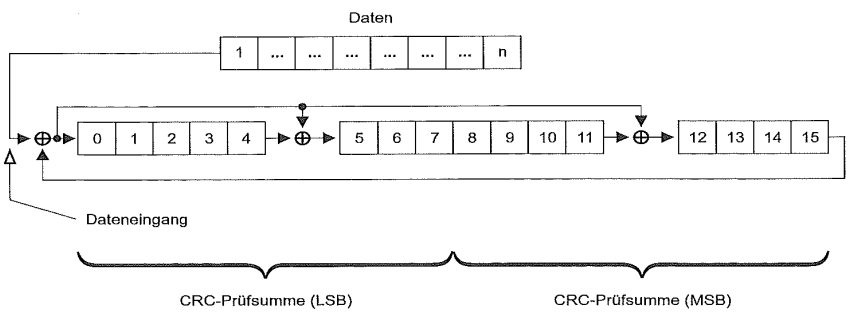


Bild 4.14 Die Berechnung einer CRC-16-Prüfsumme mit dem Generatorpolynom $G(x) = x^{16} + x^{12} + x^5 + 1$. Die Daten und das CRC-Register sind einheitlich als Bitwerte dargestellt.

Bei der Berechnung der CRC-Prüfsumme in Bild 4.14 geht man folgendermaßen vor: Zunächst setzt man das 16 Bit lange CRC-Register auf seinen Startwert. Danach beginnt man die Berechnung, indem die Datenbits, angefangen von den niederwertigsten, nacheinander in das rückgekoppelte Schieberegister gebracht werden. Die Rückkopplung bzw. die Polynomdivision gründet auf der logischen bitweisen XOR-Verknüpfung zwischen den

CRC-Bits. Nachdem alle Daten in das Register geschoben wurden, ist die Berechnung abgeschlossen, und der Inhalt der 16 Bit stellt die gewünschte CRC-Prüfsumme dar.

Die Prüfung von Daten mit CRC-Prüfsumme findet statt, indem man eine neue CRC-Prüfsumme über die Daten errechnet und dann die erhaltene Prüfsumme mit der ursprünglichen vergleicht. Sind beide identisch, folgt daraus, dass die Daten und die Prüfsumme nicht verändert wurden.

Der große Vorteil von CRC-Prüfsummen ist die Sicherheit der Fehlererkennung auch von Mehrfachfehlern, die sich nur mit sehr wenigen Verfahren erreichen lässt. Auch ist es mit einer CRC-Prüfung im Gegensatz zum XOR-Verfahren möglich, vertauschte Bytes in den Daten zu erkennen, da bei der Bildung der CRC-Prüfsumme die Reihenfolge der Datenbytes durch das rückgekoppelte Schieberegister sehr wohl eine Rolle spielt. Eine genaue Angabe der Erkennungswahrscheinlichkeiten dazu ist aber sehr schwierig, da die Detektionsmöglichkeiten stark von den Positionen der Fehler in den jeweiligen Bytes abhängen.

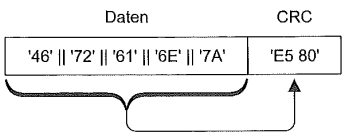


Bild 4.15 Prinzip und Rechenbeispiel der Bildung einer CRC-Prüfsumme mit dem Generatorpolynom $G(x) = x^{16} + x^{12} + x^5 + 1$ und dem Startwert '0000'.

Der Algorithmus des CRC-Verfahrens ist relativ einfach, und damit entspricht auch der Codeumfang für die Implementation den Bedürfnissen der kleinen Chipkartenspeicher. Der größte Nachteil ist jedoch die Langsamkeit der Berechnung. Dadurch, dass Bit für Bit der Daten durch den Algorithmus geschoben werden muss, sinkt die Rechengeschwindigkeit erheblich. Die CRC-Prüfsummen wurden ursprünglich für Hardware-Implementierung ausgelegt, und genau dies wirkt sich bei einer Implementierung in Software sehr nachteilig aus. Der Durchsatz einer CRC-16-Routine liegt um den Faktor 200 niedriger als der einer XOR-Prüfsumme. Ein typischer Wert dafür ist 0,2 ms/Byte bei 3,5 MHz Taktfrequenz, was einem Durchsatz von 5000 Bytes/s entspricht. Die Berechnung einer CRC-Prüfsumme über 10 kByte ROM-Speicher eines Chipkarten-Mikrocontrollers dauert demzufolge etwa 2 Sekunden.

Viele Mikrocontroller besitzen eine spezielle Komponente zur Hardware-unterstützten Bildung von CRC-Prüfsummen über festlegbare Speicherbereiche. Die dabei erzielten Geschwindigkeiten bewegen sich zum Teil in der Größenordnung von 1 Byte/Taktzyklus. Dies würde beispielsweise bei einem Mikrocontroller bei 5 MHz Takt und ohne internen Taktteiler einen Durchsatz von 5 MByte/s ergeben.

4.5.3 Reed-Solomon-Codes

Die beiden Mathematiker Irving S. Reed und Gustave Solomon veröffentlichten 1960 einen Artikel mit dem Namen „Polynomial Codes over Certain Finite Fields" und legten damit den Grundstein für eines des mittlerweile verbreitetsten Verfahren zur Fehlererkennung und Fehlerkorrektur. Die nach ihren beiden Erfindern benannten Reed-Solomon-Codes (RS-

Codes) werden zur Fehlererkennung und -korrektur bei Datenspeicherung (z. B. bei Barcode, DAT, CD, DVD) und Datenübertragung (z. B. DSL, Satelliten, Raumsonden) benutzt.

Reed-Solomon-Codes sind blockorientierte Fehlerkorrekturcodes (ECC), die auch Bündelfehler (*burst error*) korrigieren können. Ihre Berechnung beruht auf der Arithmetik von endlichen Körpern (Galois-Feldern-GF). Je nach Einsatzweck können durch bestimmte Generatorpolynome die Eigenschaften im Sinne von Anzahl korrigierbarer und erkennbarer Fehler für eine bestimmte Datenlänge bei den Reed-Solomon-Codes angepasst werden.

Seit einigen Jahren werden bei verschiedenen Chipkarten-Betriebssystemen Reed-Solomon-Codes für Fehlererkennung und sehr selten Fehlerkorrektur zur Sicherung der Datenspeicherung im EEPROM benutzt. Die verwendeten Generatorpolynome sind auf die Eigenschaften (Auftreten von Bündelfehlern) der EEPROM-Speicher abgestimmt, sodass eine deutlich sicherere Fehlererkennung als mit CRC-Verfahren möglich ist.

Beispielsweise kann man mit einem RS-Code über GF 2^8 durch 2 zusätzliche Bytes an den zu schützenden Daten 2 falsche Bytes erkennen oder 1 falsches Byte korrigieren. Mit 3 zusätzlichen Bytes können 3 falsche Bytes erkannt oder 1 falsches Byte korrigiert werden. Mit 4 zusätzlichen Bytes ist es möglich, 4 falsche Bytes zu erkennen oder 2 falsche Bytes zu korrigieren. Die Programmcode-Größe in 8051 Assembler beträgt dabei ca. 100 Byte und die Rechengeschwindigkeit ca. 10 ms/kByte bei 3,57 MHz. RS-Codes sind im Übrigen auch sehr gut in Hardware realisierbar.

4.5.4 Fehlerkorrektur

Müssen innerhalb von Speicherbereichen nicht nur Veränderungen erkannt, sondern im Fehlerfall gegebenenfalls auch noch korrigiert werden, dann ist es notwendig, Fehlerkorrekturcodes einzusetzen. Da die Berechnung dieser Codes aber aufwendig ist, also viel Programmcode benötigt, ist es problematisch, sie zur Sicherung von Chipkartenspeichern einzusetzen. Zusätzlich sind die betreffenden Algorithmen meist dafür ausgelegt, nur geringe Fehlerraten zu korrigieren. Dadurch, dass EEPROM-Speicher in Chipkarten seitenorientiert aufgebaut sind und im Fehlerfall meist eine ganze Seite ausfällt, wären nur Verfahren sinnvoll, die zusätzlich auch noch Bündelfehler korrigieren könnten. Deshalb geht man zur Fehlerkorrektur andere Wege.

Die technisch einfachste Lösung ist die Mehrfachablage der zu schützenden Daten auf physikalisch getrennten Speicherseiten mit einer Mehrheitsentscheidung beim Lesen. Üblich ist dabei eine Dreifachablage mit einer nachfolgenden 2-aus-3-Entscheidung. Eine weniger speicherplatzintensive Variante dieses Verfahrens ist die zweifache Speicherung der Daten mit jeweils zusätzlicher Absicherung durch eine EDC-Prüfsumme. Das Auftreten eines Fehlers im Speicher kann dann durch die Prüfung der beiden EDC-Werte erkannt werden. Gleichzeitig lässt sich damit auch entscheiden, in welchem der beiden getrennten Speicher der Fehler aufgetreten ist. Der zweite Speicher enthält dann die richtigen Daten, die damit restauriert werden können.

Der Mehrbedarf an Speicher bei diesen Fehlerkorrekturverfahren ist zwar erheblich, doch für kleinere Datenmengen durchaus im Rahmen des Vertretbaren. Der große Vorteil ist, dass man zur Auswertung keinen aufwendigen und programmcodeintensiven Algorithmus benötigt. Als Alternative zum obigen Schutz mittels Mehrfachablage können jedoch auch Fehlerkorrekturalgorithmen wie beispielsweise das Reed-Solomon-Verfahren

benutzt werden. Dieses ist besonders für die bei Chipkarten durch ausfallende EEPROM-Pages manchmal auftretenden Bündelfehler geeignet. Der dafür notwendige Platz an Programmcode beträgt einige hundert Byte, programmiert in Assembler, und die Größe der Daten des ECCs hängt in erster Linie davon ab, mit welcher Wahrscheinlichkeit Fehler in den Daten erkannt werden müssen bzw. korrekt korrigiert werden können.

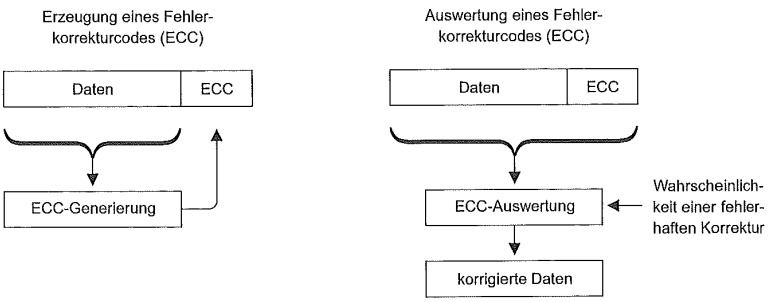


Bild 4.16 Prinzipieller Ablauf beim Einsatz eines fehlerkorrigierenden Codes (*error correction code – ECC*).

Allerdings sind zum Einsatz von Fehlerkorrekturverfahren in Chipkarten einige grundsätzliche Anmerkungen zu machen. Es erscheint auf den ersten Blick verlockend, diese Verfahren einzusetzen, da man damit auftretende Fehler im EEPROM beseitigen kann. Man erkauft sich diese vermeintliche Datensicherheit aber mit einigen schwerwiegenden Nachteilen. Der dafür notwendige Speicherplatz ist enorm. Ebenfalls steigt die Zeit beim Schreiben von Daten in den Speicher erheblich an, da sie mehrfach abgelegt werden müssen. Algorithmen, die Bündelfehler in einer Größenordnung korrigieren können, wie sie bei den seitenorientierten Speichern von EEPROMs auftreten, sind aufwendig und benötigen ebenfalls viel Speicher für die EDC-Codes. Der wesentliche Nachteil wiegt aber noch schwerer. Auch bei Fehlerkorrekturalgorithmen können prinzipiell Fehler bei der Korrektur auftreten, da sie nur bis zu einer bestimmten Anzahl von Fehlern korrekt arbeiten. Korrigiert nun ein Betriebssystem auftretende Speicherfehler automatisch, so ist im Prinzip nie sichergestellt, ob die Korrektur in richtiger Weise geschehen ist.

Man stelle sich dazu beispielsweise eine automatische Fehlerkorrektur des Saldos in einer elektronischen Geldbörse vor. Der Systembetreiber kann nie sicher sein, was mit den geladenen Beträgen im Fehlerfall passiert. Der Börseninhalt könnte richtig korrigiert werden, doch besteht auch eine gewisse Wahrscheinlichkeit, dass er nach der Korrektur zu hoch oder zu niedrig ist. Man sollte sich bei dieser Thematik dabei auch in Erinnerung rufen, dass Chipkarten preisgünstige Massenartikel sind, die man beim Auftreten von Fehlern einfach austauscht.

Bei Problemen mit den Dateninhalten muss in der Regel von einem übergeordneten System mit menschlichen Eingriffsmöglichkeiten entschieden werden, was zu tun ist. Zum Beispiel wird man manuell beim erstmaligen Auftreten eines Fehlers in einer Chipkarten-Geldbörse dem Kartenbesitzer sicherlich sein Guthaben ersetzen. Tritt der Fehlerfall aber wiederholt auf, so wird man die Kulanz bei dieser Person einschränken, da eine Betrugsabsicht durch Manipulationsversuche des EEPROMs vorliegen könnte. Dies ist nicht durch

Fehlerkorrekturcodes in der Chipkarte zu reglementieren. In diesem Fall muss ein Systemadministrator eingreifen.

4.6 Datenkompression

Der auf Chipkarten-Mikrocontrollern zur Verfügung stehende Speicherplatz ist bekanntermaßen stark limitiert, weshalb bei Anwendern immer wieder der Wunsch auftaucht, durch Datenkomprimierung eine Verbesserung zu erreichen.

Dabei gibt es allerdings einige Hindernisse zu überwinden. Der Algorithmus darf nicht allzu programmcodeaufwendig sein und sollte vor allem wenig RAM benötigen. Weiterhin sollte die Kompressionsrate einen akzeptablen Wert aufweisen. Der Kompressionsdurchsatz ist nicht von so großer Bedeutung, da die zu packenden Datenmengen in jedem Fall maximal einige hundert Bytes betragen.

Für Chipkarten kommen ausschließlich verlustfreie Kompressionsverfahren in Betracht, da die entkomprimierten Daten vollständig den Originaldaten entsprechen müssen. Die bei Chipkarten hin und wieder zur Anwendung kommenden Verfahren sind die Lauflängencodierung (*run length encoding*) und Codierungen mit variabler Länge (*encoding with variable length*).

Bei der Lauflängencodierung werden mehrfach hintereinander vorkommende Daten durch die Anzahl der Wiederholungen und das zu wiederholende Zeichen ersetzt. Bei Codierungen mit variabler Länge werden Zeichen mit fester Länge (z. B. ein Byte) in ihrer Häufigkeit analysiert und durch die am häufigsten vorkommenden Zeichen durch Zeichen mit kürzerer Länge ersetzt (Huffman-Algorithmus). Für selten vorkommende Zeichen werden dann längere Codierungen benutzt.

Bei der statischen Codierung mit variabler Länge werden die Ersetzungen anhand einer vorab fest eingestellten Tabelle durchgeführt. Die dynamische Variante dieser Codierung führt zuerst eine Analyse der Häufigkeitsverteilung der Originalzeichen durch, und auf dieser Grundlage wird dann eine Ersetzungstabelle erstellt. Als dritte Variante gibt es noch die adaptive Codierung variabler Länge, bei der während des Komprimiervorgangs laufend die Ersetzungstabelle auf die optimalen Werte eingestellt wird.

Sowohl die dynamische als auch die adaptive Codierung variabler Länge kommen aufgrund des aufwendigen Algorithmus und des großen RAM-Speicherbedarfs für Chipkarten nicht in Frage. So bleiben in der Realität nur die Lauflängencodierung und die statische Codierung variabler Länge für die Implementation in Chipkarten übrig. Der Algorithmus für die Lauflängencodierung benötigt wenig Programmcode, hat aber den Nachteil, dass er nur für sich oft hintereinander wiederholende Daten angewendet werden kann. Gut geeignet sind hier beispielsweise Bilddaten, da diese oft große einheitliche Flächen aufweisen. Schlüssel für symmetrische kryptografische Algorithmen wären zur Komprimierung mit diesem Verfahren völlig ungeeignet, da sie den Charakter von Zufallszahlen haben.

Die statische Codierung variabler Länge ist das zweite bei Chipkarten benutzte Kompressionsverfahren. Es lässt sich beispielsweise sehr gut für Dateien mit Telefonverzeichnissen einsetzen, da die dort abgelegten Daten in ihrer Struktur bekannt sind und damit die Ersetzungstabelle fest im Algorithmus verankert werden kann. So bestehen Telefonnummern nur aus den Zahlen 0 bis 9 und einigen wenigen Sonderzeichen, wie „#“ und „*“. Sind für die Namen nur Großbuchstaben erlaubt, müssen nur noch die 26 Zeichen des Al-

phabets in der Ersetzungstabelle berücksichtigt werden. Ebenfalls werden bestimmte Buchstaben bei Namen deutlich seltener vorkommen als andere, was ebenfalls Einfluss auf die Codierung hat. Bei Telefonverzeichnissen kann man auf diese Weise durchaus eine Ersparnis von 30 Prozent Speicherplatz gegenüber der unkomprimierten Variante erreichen. Der Programmspeicherbedarf für den Komprimieralgorithmus ist dabei jedoch noch nicht berücksichtigt.

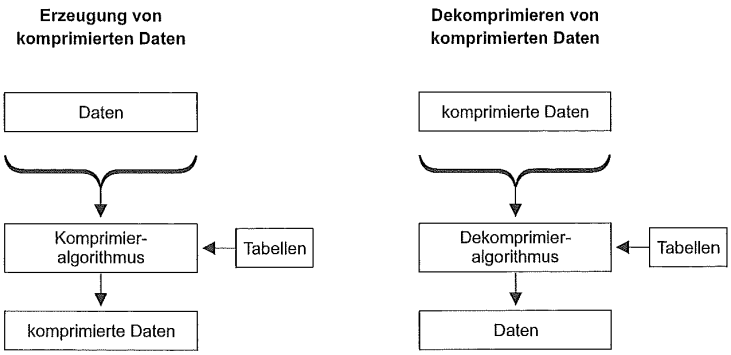


Bild 4.17 Prinzipieller Ablauf bei der Datenkomprimierung von zu speichernden Daten.

Bei der Datenkomprimierung in Chipkarten ist jedoch einiges zu beachten. Idealerweise findet sie im Betriebssystem für die äußere Welt völlig transparent statt, sodass mit den Standardkommandos auf übliche Weise die unkomprimierten Daten geschrieben und gelesen werden können. Die Komprimierung kann auch nur auf bestimmte Arten von Daten angewendet werden. So sind sowohl Assemblercode als auch kryptografische Schlüssel in der Regel nicht zufriedenstellend komprimierbar. Dies muss beim Anwendungsdesign berücksichtigt werden, da sonst die vermeintliche Ersparnis von Speicherplatz im ungünstigsten Fall durch die Komprimierung sogar einen Mehrbedarf an Speicher verursachen kann.

Diese Gründe sind die Ursache, warum Datenkomprimierung für Chipkarten-Dateien bisher nur verhalten eingesetzt wird. In speziellen Anwendungen, wie etwa den Telefonverzeichnissen bei Karten des Telekommunikationssektors, wird manchmal mit Komprimieralgorithmen gearbeitet. Bei allgemeinen Betriebssystemen und Anwendungen, bei denen die Struktur der Daten nicht im Voraus bekannt ist, führt Datenkompression zu keinem befriedigendem Ergebnis und sollte aufgrund des zusätzlichen Speicherbedarfs für den Komprimieralgorithmus auch vermieden werden.

4.7 Kryptologie

Neben der Funktion als Datenspeicher sind Chipkarten auch Berechtigungsträger und Verschlüsselungsmodule. Dies führte schon in den Anfängen der Chipkarten-Entwicklung dazu, dass die Kryptologie eine zentrale Bedeutung gewann. Mittlerweile gehören die Verfahren und Methoden dieser Wissenschaftsdisziplin fest zur Chipkartentechnik.

Die Kryptologie teilt sich in die beiden Tätigkeitsfelder Kryptografie und Kryptoanalyse auf. Als Kryptografie bezeichnet man die Wissenschaft von den Methoden der Ver- und Entschlüsselung von Daten. Die Kryptoanalyse als Wissenschaft versucht bestehende kryptografische Systeme zu brechen.

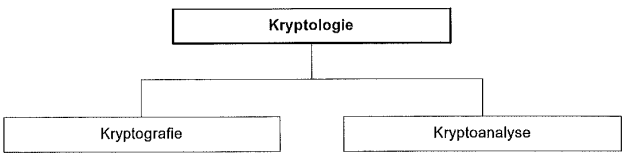


Bild 4.18 Klassifizierungsbaum der beiden Tätigkeitsfelder der Kryptologie: Kryptografie und Kryptoanalyse.

Im Chipkartenbereich ist vor allem die praktische Anwendung von bestehenden kryptografischen Verfahren und Methoden die zentrale Aufgabenstellung und das Haupteinsatzgebiet. Deshalb sind hier mehr die praktischen als die theoretischen Aspekte der Kryptologie behandelt. Jedoch ist auch Wert auf die Anwendung der Verfahren und auf die Grundzüge des theoretischen Hintergrundes gelegt.

Die vier Ziele der Kryptologie sind die Geheimhaltung von Nachrichten (*confidentiality*), die Sicherstellung von Integrität (*integrity*) und Authentizität (*authenticity*) von Nachrichten und die Verbindlichkeit (*non-repudiation*) von Nachrichten. Die Ziele sind unabhängig voneinander und stellen auch unterschiedliche Anforderungen an das jeweilige System. Geheimhaltung bedeutet, dass nur der oder die gewünschten Empfänger den Inhalt einer Nachricht entschlüsseln können. Kann der Empfänger sicherstellen, dass die erhaltene Nachricht während der Übertragung nicht verändert wurde, dann bezeichnet man dies als Authentizität. Hat der Absender die Möglichkeit zu prüfen, ob ein bestimmter Empfänger eine Nachricht erhalten hat, so nennt man dies Verbindlichkeit oder Nichtabstreitbarkeit einer Nachricht.

Die in diesem Buch verwendeten Notationen für kryptografische Verfahren sind in Bild 4.18 dargestellt. Die folgenden Termini und Prinzipien stellen eine Grundlage der Kryptologie dar und sind zum Verständnis der vorgestellten Verfahren Voraussetzung.

Vereinfacht kennt die Verschlüsselungstechnik drei Arten von Daten. Als Klartext (*plaintext*) werden unverschlüsselte Daten bezeichnet. Im Gegensatz dazu nennt man die

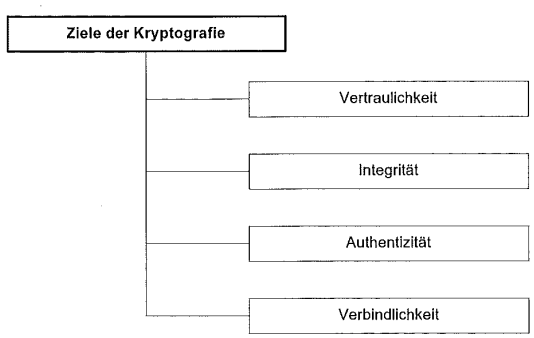


Bild 4.19 Klassifizierungsbaum der voneinander unabhängigen Ziele der Kryptografie.

verschlüsselten Daten den Schlüsseltext (*ciphertext*). Zur Ver- und Entschlüsselung werden ein oder mehrere Schlüssel (*key*) benötigt, welche die dritte Datenart sind. Diese drei Datenarten werden von einem Verschlüsselungsalgorithmus verarbeitet. In Chipkarten eingesetzte Kryptoalgorithmen sind momentan durchgängig blockorientiert. Dies bedeutet, dass Klar- und Schlüsseltext immer nur in Paketen mit einer festen Länge (z. B. 8 Byte beim DES) verarbeitet werden können.

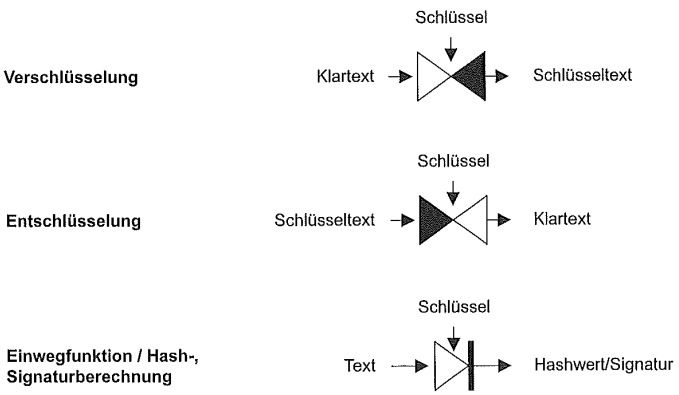


Bild 4.20 Die in diesem Buch verwendete Symbolik für kryptografische Algorithmen.

Moderne Kryptoalgorithmen basieren in der Regel auf dem Kerckhoff-Prinzip. Das nach Auguste Kerckhoff (1835–1903) benannte Prinzip besagt, dass die gesamte Sicherheit eines Algorithmus nur auf der Geheimhaltung des Schlüssels beruhen soll und nicht auf der Geheimhaltung des kryptografischen Algorithmus. Die Auswirkung dieses allgemein anerkannten, aber oft missachteten Prinzips war, dass viele im zivilen Bereich eingesetzte Kryptoalgorithmen veröffentlicht wurden und teilweise auch genormt sind.

Der Gegensatz zu Kerckhoff ist das Prinzip der Sicherheit durch Verschleiern. Die Sicherheit eines Systems beruht bei diesem Prinzip darauf, dass ein fiktiver Angreifer nicht weiß, wie das System funktioniert. Dieses Prinzip ist sehr alt und wird auch heute noch oft angewendet. Man sollte sich jedoch hüten, alleinig nach diesem Prinzip ein Kryptosystem oder auch ein anderes System zu entwickeln. Bisher wurden noch alle Systeme, die nur darauf beruhten, gebrochen, und dies meist auch noch in sehr kurzer Zeit. Es ist in unserer Informationsgesellschaft im Allgemeinen nicht mehr möglich, technische Details eines Systems über längere Zeit geheim zu halten, und dies ist genau der Knackpunkt bei diesem Prinzip.

Allerdings lassen sich durch Verschleierung die Auswirkungen von zufälligem Abhören von Nachrichten sehr wohl reduzieren. Deshalb findet dieses Prinzip parallel neben dem von Kerckhoff immer wieder Anwendung. In vielen großen Systemen wird es auch noch als eine zusätzliche Sicherheitsstufe eingebaut. Da die Sicherheit moderner und veröffentlichter Kryptoalgorithmen vor allem auf der begrenzten Rechenleistung heutiger Computern beruht, erreicht man durch zusätzliche Verschleierung der Verfahren eine zusätzliche Steigerung des Schutzes vor Angriffen.

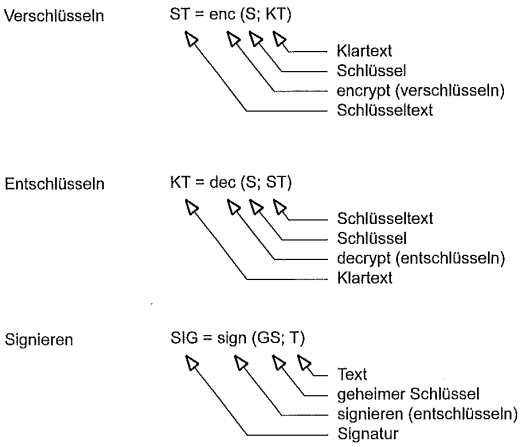


Bild 4.21 Die in diesem Buch verwendeten Notationen für kryptografische Verfahren.

Verlässt man sich nur auf den Schutz einer dem vermeintlichen Angreifer nicht zur Verfügung stehenden Rechenleistung, so kann man unter Umständen sehr schnell von der rapide fortschreitenden technischen Weiterentwicklung überrollt werden. Angaben wie „man braucht tausend Jahre, um dieses Kryptosystem zu brechen" stellen keine verlässliche Aussage dar, da sie von heute zur Verfügung stehenden Rechenleistungen und Algorithmen ausgehen und zukünftige Entwicklungen nicht berücksichtigen können, da diese im Allgemeinen auch nicht bekannt sind. Etwa alle 18 Monate verdoppelt sich die Rechenleistung von Prozessoren, was dazu führte, dass sich seit den letzten 25 Jahren die pro Prozessor zu Verfügung stehende Rechenleistung um etwa den Faktor 25 000 vergrößert hat.

In den letzten Jahren ist durch die zunehmende Vernetzung von Computern eine weitere Möglichkeit geschaffen worden, ernst zu nehmende Angriffe auf Schlüssel oder Kryptosysteme durchzuführen. Man denke nur an eine Aufforderung zum Brechen eines DES-Schlüssels im Internet, die im Schneeballsystem an die Millionen von Benutzer weitergeleitet wird. Sollte sich nur ein Prozent aller derzeitigen[2] Teilnehmer an einer solchen Aktion beteiligen, so stünde dem potenziellen Angreifer ein aus 300 000 individuellen Computern bestehender Parallelrechner zur Verfügung.

Kryptoalgorithmen teilt man in symmetrische und asymmetrische Algorithmen ein. Diese Einteilung bezieht sich auf die verwendeten Schlüssel. Symmetrisch sagt dabei aus, dass der Algorithmus für Ver- und Entschlüsselung den gleichen Schlüssel benutzt. Im Gegensatz dazu benötigen die im Jahre 1976 von Whitfield Diffie und Martin E. Hellman postulierten asymmetrischen Kryptoalgorithmen für Ver- und Entschlüsselung je einen unterschiedlichen Schlüssel.

Ein Begriff kommt im Zusammenhang mit Kryptoalgorithmen öfters vor. Es ist dies die Mächtigkeit des Schlüsselraums. Damit bezeichnet man die Anzahl der möglichen Schlüssel für einen Kryptoalgorithmus. Ein großer Schlüsselraum ist eines von mehreren Kriterien für einen sicheren Kryptoalgorithmus.

[2] Im Sommer 2001 wurde davon ausgegangen, dass im Internet etwa 500 Millionen Teilnehmer existieren.

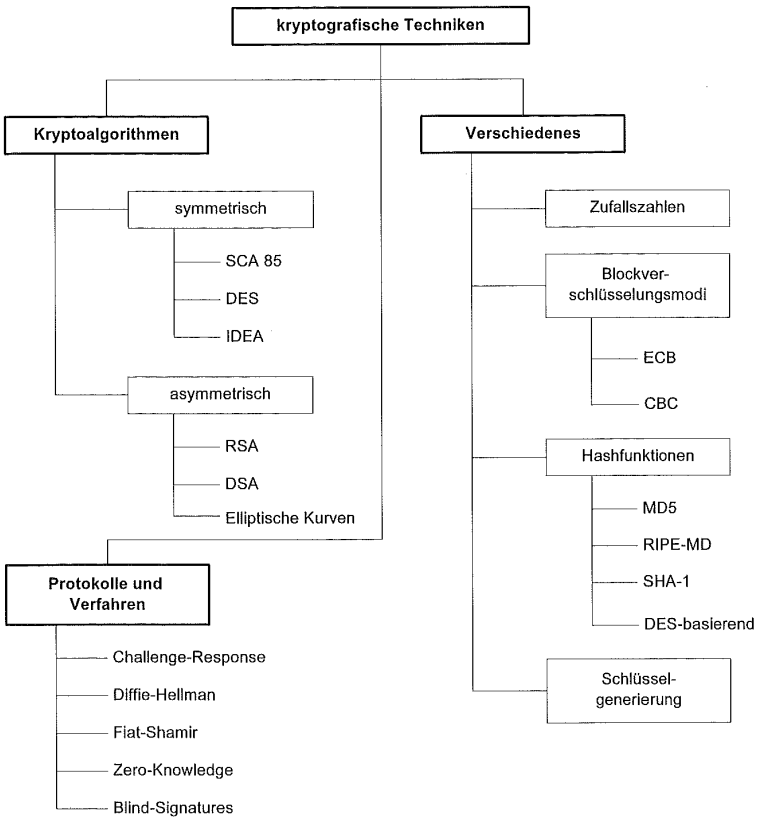


Bild 4.22 Klassifizierungsbaum der im Umfeld von Chipkarten eingesetzten kryptografischen Techniken.

Eine erst seit kürzerer Zeit sehr in den Vordergrund gerückte Forderung für die technische Realisation von kryptografischen Algorithmen in Chipkarten ist die Rauschfreiheit. Der Begriff bedeutet in diesem Zusammenhang, dass die Ausführungszeit des Algorithmus unabhängig vom Schlüssel, Klartext bzw. Schlüsseltext sein muss. Ist diese Forderung nicht erfüllt, kann dies dazu führen, dass der geheime Schlüssel innerhalb sehr kurzer Zeit ermittelt und das gesamte kryptografische System gebrochen werden kann.

Die Kryptologie unterscheidet streng nach theoretischer und praktischer Sicherheit eines Systems oder Algorithmus. Ein System ist theoretisch sicher, wenn einem Angreifer unbegrenzte Zeit und Hilfsmittel zur Verfügung stehen und es ihm selbst dann nicht möglich ist, das System zu brechen. Dies bedeutet beispielsweise, dass ein System auch dann nicht mehr als theoretisch sicher bezeichnet wird, wenn ein Angreifer Hunderte von Jahren und mehrere Supercomputer zum Brechen benötigen würde. Stehen dem Angreifer nur begrenzte Zeit und Hilfsmittel zur Verfügung und kann er damit das System nicht brechen, dann bezeichnet man das System als praktisch sicher.

Ein kryptografisches System kann Geheimhaltung und/oder Authentizität einer Nachricht sicherstellen. Dieses System zu brechen bedeutet, dass die Geheimhaltung und/oder die Authentizität gegenüber einem Angreifer nicht mehr gewährleistet ist. Wenn der Angreifer beispielsweise den geheimen Schlüssel eines Verschlüsselungsalgorithmus herausfindet, dann ist es ihm möglich, die verschlüsselten und damit geschützten Daten zu entschlüsseln, den Inhalt der Daten zu erfahren und sie auch bei Bedarf zu ändern.

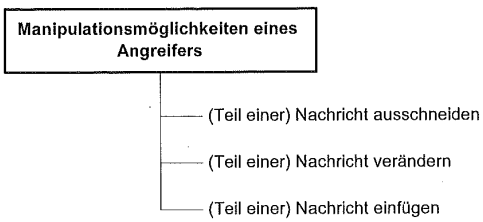


Bild 4.23 Klassifizierungsbaum der Manipulationsmöglichkeiten eines Angreifers.

Um den Schlüssel eines Kryptoalgorithmus zu brechen, gibt es verschiedene Angriffsmöglichkeiten. Bei der „ciphertext only attack" kennt der Angreifer nur den Schlüsseltext und versucht, mit dieser Information Schlüssel oder Klartext herauszufinden. Der in Hinblick auf einen Erfolg aussichtsreichere Angriff, die „known plaintext attack" bedeutet, dass der Angreifer mehrere Klartext-Schlüsseltext-Paare für einen geheimen Schlüssel kennt. Kann der Angreifer in der „chosen plaintext attack" und der „chosen ciphertext attack" eigene Klartext- bzw. Schlüsseltext-Paare generieren, dann ist dies einer der aussichtsreichsten Angriffe. Denn dann kann durch Probieren der geheime Schlüssel gefunden werden.

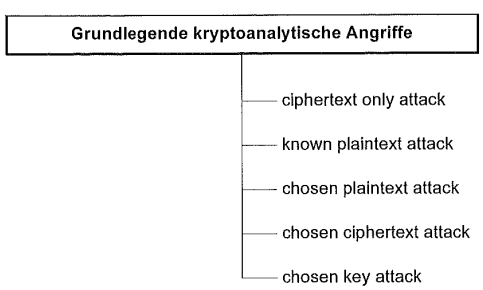


Bild 4.24 Klassifizierungsbaum der grundlegenden kryptoanalytischen Angriffe.

Das Finden des Schlüssels durch Ausprobieren (*brute force attack*) ist natürlich der trivialste Angriff. Dabei versucht man, mit großer Rechenleistung durch Probieren den richtigen Schlüssel herauszufinden. Auf der Grundlage eines bekannten Klartext-Schlüsseltext-Paares probiert man alle möglichen Schlüssel aus. Dass dabei üblicherweise Rechenleistungen im Bereich von Supercomputern Voraussetzung sind, versteht sich von selbst. Statistisch gesehen muss man im Mittel die Hälfte aller möglichen Schlüssel durchprobieren, um den richtigen zu finden. Ein großer Schlüsselraum erschwert diesen Angriff allerdings erheblich.

4.7.1 Symmetrische Kryptoalgorithmen

Die symmetrischen Kryptoalgorithmen basieren auf dem Prinzip, die Ver- und Entschlüsselung mit dem gleichen Schlüssel durchzuführen. Daher die Bezeichnung symmetrisch.

DES-Algorithmus

Der bekannteste und verbreitetste Vertreter ist der Data Encryption Algorithm, kurz DEA genannt. Dieser Algorithmus wurde von IBM zusammen mit dem NBS (US National Bureau of Standards) entwickelt und 1977 als US-Norm (FIPS 46) publiziert. Die Norm, die den DEA beschreibt, wird in der Regel als DES (*data encryption standard*) bezeichnet. Aus diesem Grund wird der Data Encryption Algorithm oft, wenn auch nicht ganz korrekt, DES genannt.

Da dieser Algorithmus natürlich nach Kerckhoffs Prinzip gestaltet ist, konnte er auch ohne Einbußen an Sicherheit veröffentlicht werden. Allerdings wurden bis heute nicht alle Entwicklungskriterien bekannt gegeben, was immer wieder zu Vermutungen bezüglich der Angriffsmöglichkeiten und vorhandenen Falltüren führt. Jedoch sind bis jetzt alle Versuche, den Algorithmus auf diese Weise zu brechen, fehlgeschlagen.

Zwei wichtige Prinzipien für einen guten Verschlüsselungsalgorithmus gingen in das Design des DES ein. Die beiden Prinzipien sind Konfusion und Diffusion nach C. Shannon. Das Prinzip der Konfusion sagt aus, dass die Statistik des Schlüsseltextes die Statistik des Klartextes in einer solch komplexen Weise beeinflussen soll, dass ein Angreifer daraus keine Vorteile erlangen kann. Das zweite Prinzip, die Diffusion, besagt, dass jedes Bit des Klartextes und des Schlüssels möglichst viele Bits des Schlüsseltextes beeinflussen soll.

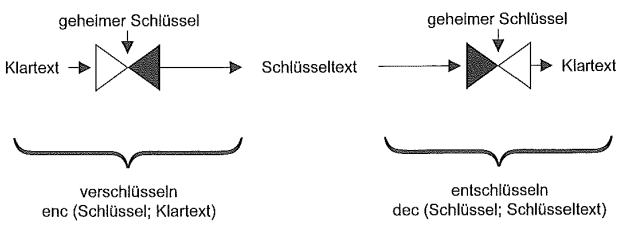


Bild 4.25 Das Prinzip eines symmetrischen Kryptoalgorithmus bei der Verschlüsselung und darauf folgender Entschlüsselung. Der DES ist das typische Beispiel für diese Art von kryptografischem Algorithmus.

Der symmetrische Blockverschlüsselungsalgorithmus DES führt keine Expansion des Schlüsseltextes durch, was bedeutet, dass Klartext- und Schlüsseltextblöcke gleich lang sind. Die Blocklänge beträgt 64 Bit (= 8 Byte), ebenso wie die Schlüssellänge, allerdings werden nur 56 Bit davon als Schlüssel benutzt.

Der Schlüssel enthält 8 Paritätsbits, was den verfügbaren Schlüsselraum reduziert. Die 64 Bit eines Schlüssels sind von links (msb) nach rechts (lsb) durchnummeriert, und die Bits 8, 16, 24, ..., 64 stellen das Paritätsbit dar. Die Parität ist dabei immer ungerade (*odd*). Der Schlüsselraum beträgt beim DES aufgrund dieser 8 Paritätsbits 2^{56}. Damit ergeben sich $2^{56} \approx 7{,}2 \cdot 10^{16}$ Möglichkeiten für einen Schlüssel. Auf den ersten Blick mag dieser

Schlüsselraum mit der Möglichkeit von 72 057 594 037 927 936 Schlüsseln als sehr groß erscheinen, doch dies ist der Hauptschwachpunkt des DES.[3] Angesichts der stetig steigenden Rechenleistung moderner Computer gilt ein Schlüsselraum bereits als zu klein für einen sicheren Kryptoalgorithmus. Bei einem zu kleinen Schlüsselraum können mit einem vorhandenen Klartext-Schlüsseltext-Paar einfach alle Möglichkeiten durchprobiert werden, um den geheimen Schlüssel zu ermitteln.

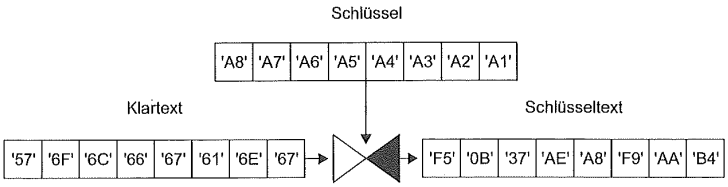


Bild 4.26 Die Funktionalität des DES-Algorithmus bei einer Verschlüsselungsoperation.

Bekommt man durch Abhorchen der Kommunikation zwischen Terminal und Chipkarte ein Klartext-Schlüsseltext-Paar, dann kann man den folgenden Brute-force-Angriff ausführen. Man verschlüsselt mit allen möglichen Schlüsseln den Klartext. Durch Vergleich des Ergebnisses mit dem vorgegebenen Schlüsseltext lässt sich feststellen, ob man den richtigen Schlüssel gefunden hat. Diese Vorgehensweise kann man hervorragend parallelisieren. Jeder der einzelnen Parallelrechner probiert für sich alleine einen kleinen Teil des Schlüsselraumes durch. Die folgende Beispielrechnung verdeutlicht den Zeitaufwand für diesen Brute-force-Angriff. Die zurzeit schnellsten DES-Bausteine benötigen für eine komplette Blockverschlüsselung 64 ns.[4] Baut man nun 10000 dieser Recheneinheiten parallel auf, dann können diese unabhängig voneinander jeder für sich einen kleinen Teil des Schlüsselraumes prüfen. Unter der Annahme, dass durchschnittlich nur der halbe Schlüsselraum durchsucht werden muss, um den richtigen Schlüssel zu finden, ist folgende Rechnung möglich:

$$\frac{2^{56} \cdot 64\,\text{nsec}}{10\,000} \cdot \frac{1}{2} \approx 64\,\text{h}\,.$$

Michael Wiener veröffentlichte 1993 die Pläne für einen eine Million Dollar teuren Computer, der innerhalb von sieben Stunden alle DES-Schlüssel eines gegebenen Klartext-Schlüsseltext-Paars durchprobieren konnte [Wiener 93]. Von der EFF (Electronic Frontier Foundation) wurde ein ähnliches Konzept namens „DES Cracker" 1998 um ca. 250000 US$ verwirklicht. Der DES Cracker kann ca. 88,8 Milliarden Schlüssel pro Sekunde prü-

[3] Um eine Vorstellung von großen Zahlen zu gewinnen, sollte man sich die folgende Abschätzung zu Gemüte führen: Das Gewicht der Erde beträgt schätzungsweise $5{,}974 \cdot 10^{27}$ g. Auf dieser – allerdings sehr ungenauen – Datengrundlage lässt sich die Anzahl der Elektronen, Protonen und Neutronen, aus der die Erde besteht, mit $\approx 10^{52}$ angeben. Würde für die Speicherung eines Bits nur ein einzelnes Elementarteilchen benötigt, so könnte man demnach mit einem Speicher, der die Gesamtmasse der Erde hat, maximal 10^{52} Bit speichern.

[4] Es handelt sich dabei um ein Gate-Array der Firma DEC in Galliumarsenid-Technologie für ECB- und CBC-Modus.

fen und hat im ersten öffentlichen Test zur Ermittlung eines unbekannten DES-Schlüssels 56 Stunden benötigt [EFF 98]. Das größte Problem bei der Verwendung des DES ist der mittlerweile zu klein gewordene Schlüsselraum, sodass bei neuen Anwendungen beinahe ausschließlich der Triple-DES benutzt wird.

Hier auf die genaue Implementation des DES einzugehen, würde den Umfang dieses Buches sprengen und ist auch zum Verständnis der Materie nicht notwendig. Dazu sei auf die FIPS Publication 46, Carl Meyer [Meyer 82] oder Bruce Schneier [Schneier 96] verwiesen. Ein Detail ist allerdings noch von Bedeutung: Der DES als Verschlüsselungsalgorithmus wurde so gestaltet, dass er sich leicht als Hardwareschaltung aufbauen lässt. Viele Chipkarten-Mikrocontroller verfügen mittlerweile über einen DES-Hardwarebaustein. Muss der DES in Chipkarten in Software realisiert werden, dann beträgt der Umfang selbst bei hochoptimierten Versionen etwa 1 kByte Assemblercode. In DPA-resistenter Version beträgt die Größe etwa 2 kByte. Dass die Berechnungsgeschwindigkeit langsam ist, ergibt sich aus Obigem zwangsläufig.

Die typischen Zeiten für Ver- und Entschlüsselung bei Chipkarten im Vergleich zu einem Hardwarebaustein und einem PC sind in der nachfolgenden Tabelle aufgeführt. Die Zeiten können je nach Implementierung differieren und berücksichtigen nur die reine Rechenzeit für eine DES-Ver- oder Entschlüsselung eines 8-Byte-Blocks unter der Voraussetzung, dass alle Register bereits vorgeladen sind. Als Richtwert kann man jedoch davon ausgehen, dass der DES in Hardware realisiert etwa 150-mal schneller ist als in Software.

Die Erzeugung eines Schlüssels für den DES-Algorithmus kann unter Benutzung eines Zufallszahlengenerators stattfinden. Dieser generiert eine 8 Byte lange Zufallszahl, die dann auf die vier schwachen und zwölf semi schwachen Schlüssel geprüft wird. Fällt sie nicht unter diese einfach zu brechenden Schlüssel, führt man noch die Paritätsberechnung durch. Das Ergebnis ist ein DES-Schlüssel.

Tabelle 4.10 Tabelle mit typischen Beispielen für die Geschwindigkeit des DES

Realisierung	Geschwindigkeit
Chipkarte mit 3,5 MHz Takt, Realisierung in Software	17,0 ms/8 Byte Block = 3,8 kBit/s
Chipkarte mit 3,5 MHz Takt und DES-Recheneinheit	112 µs/8 Byte Block = 571 kBit/s
Chipkarte mit 3,5 MHz Takt und Triple-DES-Recheneinheit	130 µs/8 Byte Block = 492 kBit/s
Chipkarte mit 4,9 MHz Takt, Realisierung in Software	12,0 ms/8 Byte Block = 5,3 kBit/s
Chipkarte mit 4,9 MHz Takt und DES-Recheneinheit	80 µs/8 Byte Block = 800 kBit/s
Chipkarte mit 4,9 MHz Takt und Triple-DES-Recheneinheit	93 µs/8 Byte Block = 688 kBit/s
PC (80 486, 33 MHz)	30 µs/8 Byte Block = 2,1 MBit/s
PC (Pentium, 90 MHz)	16 µs/8 Byte Block = 4 MBit/s
PC (Pentium, 200 MHz)	4 µs/8 Byte Block = 16 MBit/s
DES-Hardwarebaustein	64 ns/8 Byte Block = 100 MBit/s

IDEA-Algorithmus

Neben dem DES existieren noch viele weitere symmetrische Kryptoalgorithmen. Stellvertretend dafür sei hier nur noch der IDEA (*international data encryption algorithm*) genannt. Er wurde von Xuejia Lai und James L. Massey entwickelt und seit seiner Veröffent-

lichung im Jahr 1990 als PES (*proposed encryption standard*) einmal im Jahr 1991 verbessert. Der verbesserte Algorithmus erhielt damals kurzfristig den Namen IPES (*improved proposed encryption standard*) und ist aber heute allgemein als IDEA bekannt. Die Entwicklungskriterien und der interne Aufbau dieses Algorithmus sind vollständig veröffentlicht, sodass das Kerckhoff-Prinzip erfüllt ist. Allerdings unterliegt er, ähnlich wie der RSA-Algorithmus, patentrechtlichen Beschränkungen.

Der IDEA ist wie der DES ein blockorientierter Kryptoalgorithmus und benutzt ebenfalls 8 Byte Klar-/Schlüsseltext-Blöcke. Die Schlüssellänge ist im Gegensatz zum DES aber 16 Byte (= 2 · 8 Byte), sodass ein wesentlich ausgedehnterer Schlüsselraum der Größe $2^{128} \approx 3{,}4 \cdot 10^{38}$ zur Verfügung steht. In der üblichen dezimalen Schreibweise dargestellt, beträgt die Anzahl der möglichen Schlüssel beim IDEA exakt 340 282 366 920 938 463 463 374 607 431 768 211 456.

Durch den beschriebenen Aufbau ist der IDEA bis auf die vergrößerte Schlüssellänge zum DES kompatibel. Zu Triple-DES-Systemen, die 2 · 56 Bit lange Schlüssel verwenden, besteht ebenfalls eine Kompatibilität, sodass beim Wechsel des Kryptoalgorithmus keine Auswirkungen auf die Längen von Schlüsseln oder Eingangs- bzw. Ausgangsdaten auftreten. Natürlich bedeutet Kompatibilität in diesem Zusammenhang nicht, dass DES-verschlüsselte Daten mit dem IDEA entschlüsselt werden können. Im Allgemeinen wird der IDEA als ein sehr guter Kryptoalgorithmus angesehen und ist auch mittlerweile durch das Public-Domain-Programm zur Sicherung der Datenübertragungen PGP (*pretty good privacy*) von Philip Zimmermann weit verbreitet.

Es existieren wenige Implementationen des IDEA auf Chipkarten. Der Speicherbedarf für das Programm bewegt sich im Bereich von 1 000 Byte. Die typischen Rechenzeiten für Ent- und Verschlüsselung sind etwas geringer als für den DES. Bei der Entwicklung des IDEA ist man aber davon ausgegangen, dass die Berechnungen durch einen 16 Bit-Prozessor durchgeführt werden. Da Chipkarten aber immer noch vorrangig 8 Bit-Prozessoren haben, ist der Geschwindigkeitsvorteil gegenüber dem DES nicht so hoch wie erwartet. In der unten stehenden Tabelle sind zwei Beispielwerte für eine IDEA-Operation für einen 8-Byte-Block unter Zugrundelegung von vorberechneten Schlüsseln angegeben.

Tabelle 4.11 Tabelle mit typischen Beispielen für die Geschwindigkeit des IDEA

Realisierung	Geschwindigkeit
Chipkarte mit 3,5 MHz Takt, Realisierung in Software	12,3 ms/8 Byte Block
Chipkarte mit 4,9 MHz Takt, Realisierung in Software	8,8 ms/8 Byte Block
PC (80 386, 33 MHz)	70 µs/8 Byte Block
PC (Pentium Pro, 180 MHz)	4 µs/8 Byte Block
IDEA-Hardwarebaustein	370 ns/8 Byte Block

AES-Algorithmus

Die mögliche zur Verfügung stehende Rechenleistung erreichte durch die technische Entwicklung und weltweite Vernetzung von Rechnern Ende der 90er-Jahre ein Niveau, aufgrund dessen selbst für ambitionierte Privatpersonen mit Organisationstalent ein erfolgreicher brute-force-Angriff auf den DES-Algorithmus möglich wurde. Die infolgedessen im-

mer mehr in Zweifel gekommene Zukunftstauglichkeit des DES setzte die entsprechenden staatlichen Stellen immer mehr unter Zugzwang, einen neuen symmetrischen Kryptoalgorithmus als offiziellen Nachfolger des DES festzulegen. Aufgrund der nie ganz verstummten Zweifel an der Integrität des DES durch die zum Teil nicht öffentlich nachvollziehbaren Designkriterien wurde vom US-amerikanischen NIST (US National Institute of Standards and Technology [NIST]) im Jahr 1997 ein Aufruf an die weltweite Kryptologengemeinschaft getätigt, potenzielle DES-Nachfolger zu einer Art Vergleichswettbewerb beim NIST mit allen dazugehörigen Unterlagen einzureichen. 1998 verkündete das NIST dann, dass es 15 Kandidaten für einen AES (*Advanced Encryption Standard*) gibt, und veröffentlichte alle Unterlagen zur informationstechnischen und kryptoanalytischen Untersuchung durch das NSA (US National Security Agency), Forschungsinstitute, Experten und Interessierte. Nach eingehender Untersuchung der Algorithmen und breiter öffentlicher Diskussion der Ergebnisse wurde 2000 vom NIST ein Kandidat als Nachfolger des DES bekannt gegeben.[5] Er wurde von den beiden Belgischen Kryptologen Joan Daemen und Vincent Rijmen entwickelt und war bereits vor der Ausschreibung als Rijndael-Algorithmus bekannt.

Der AES ist ein symmetrischer Blockverschlüsselungsalgorithmus mit einer Blocklänge von 128 Bit (= 16 Byte) und kann mit den drei verschiedenen Schlüssellängen 128 Bit (16 Byte), 192 Bit (= 24 Byte) und 256 Bit (= 32 Byte) benutzt werden. Je nach Schlüssellänge wird der AES deshalb als AES-128, AES-192 oder AES-256 bezeichnet. Der AES lässt sich als Hardwareschaltung implementieren und kann auch in Software sowohl auf leistungsschwachen 8-Bit-Prozessoren als auch auf leistungsstarken 32-Bit- und 64-Bit-Prozessoren gut realisiert werden. Weiterhin ist er weltweit lizenzfrei benutzbar und ihm wird nach offizieller Aussage des NIST eine Lebensdauer von über 20 Jahren bescheinigt. Der AES ist in der FIPS 197 genormt, welche kostenlos via Internet [NIST] verfügbar ist.

Der Schlüsselraum des AES mit 128 Bit Schlüssel hat eine Größe von $2^{128} \approx 3{,}4 \cdot 10^{38}$ und ist damit um den Faktor $4{,}7 \cdot 10^{21}$ größer als der Schlüsselraum des DES mit 56 Bit Schlüssellänge. Aufgrund dieses großen Schlüsselraums bereits bei 128 Bit langen Schlüsseln werden Angriffe durch Durchprobieren aller möglichen Schlüssel enorm erschwert. Die softwaretechnische Realisierung des AES in DPA-resistenter Form auf einer Chipkarte benötigt ca. 4 kByte ROM.

Tabelle 4.12 Tabelle mit typischen Beispielen für die Geschwindigkeit des AES bei Verwendung von 128 Bit Schlüsseln

Realisierung	Geschwindigkeit
Chipkarte, 16 Bit CPU, 4,9 MHz Takt, Realisierung in Software, 128 Bit Schlüssellänge, verschlüsseln	20 ms/16 Byte Block
Chipkarte, 16 Bit CPU, 4,9 MHz Takt, Realisierung in Software, 128 Bit Schlüssellänge, entschlüsseln	25 ms/16 Byte Block

Betriebsarten für Blockverschlüsselungsalgorithmen

Der DES kann wie jeder Blockverschlüsselungsalgorithmus in vier verschiedenen Betriebsarten betrieben werden, welche in der ISO 8372 genormt sind. Zwei dieser Betriebsarten sind speziell für sequenzielle Texte ohne Blockstruktur geeignet (CFB, OFB-Modus).

[5] Einen guten Überblick über den Auswahlprozess und die Auswahlkriterien findet sich in [Nechvatal 00].

Die beiden anderen (ECB-, CBC-Modus) basieren auf einer Blockgröße von 8 Byte.Vor allem diese beiden blockorientierten Modi finden im Chipkartenbereich Anwendung.

Die Grundbetriebsart des DES bezeichnet man als ECB-Modus (*electronic code book*). Dabei werden 8 Byte lange Klartextblöcke unabhängig voneinander mit dem gleichen Schlüssel verschlüsselt. Dies ist der DES in seiner Reinform, also ohne Ergänzungen.

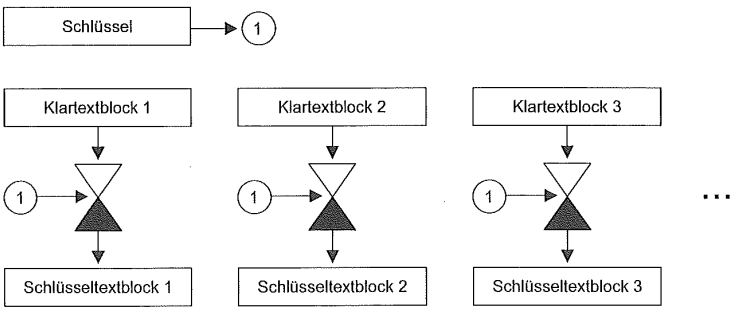


Bild 4.27 Die ECB-Betriebsart eines Blockverschlüsselungsalgorithmus bei der Verschlüsselung. Die Entschlüsselung läuft analog der Verschlüsselung ab.

Die zweite blockorientierte Betriebsart wird als CBC-Modus (*cipher block chaining*) bezeichnet. Dabei wird ein aus mehreren Blöcken bestehender Datenstring mit einer XOR-Operation bei der Verschlüsselung so verkettet, dass die nachfolgenden Blöcke von den vorhergehenden abhängig werden. Damit kann das Vertauschen, Einfügen oder Löschen von verschlüsselten Blöcken zuverlässig erkannt werden. Dies ist im ECB-Modus nicht möglich.

Bei geeigneter Gestaltung der Klartextblöcke (vorangestellter Sendefolgezähler oder Sendefolgezähler im Initialisierungsvektor) werden durch diese Verkettung selbst identische Klartextblöcke auf verschiedene Schlüsseltextblöcke abgebildet. Dies erschwert die Kryptoanalyse von abgehörten Daten ganz erheblich, da z. B. Codebuchanalysen unmöglich gemacht werden.

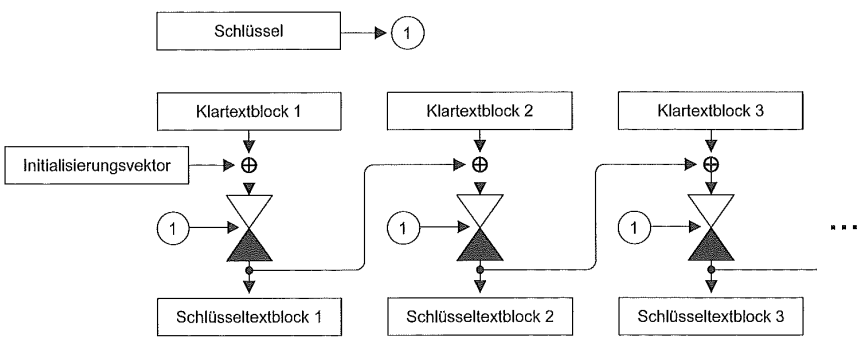


Bild 4.28 Die CBC-Betriebsart eines Blockverschlüsselungsalgorithmus bei der Verschlüsselung. Die Entschlüsselung läuft analog der Verschlüsselung ab.

Der erste Klartextblock wird mit einem Initialisierungsvektor (oft auch als IV bezeichnet) mit XOR verknüpft und dann mit dem DES verschlüsselt. Das Ergebnis ist der Schlüsseltext, der wiederum mit dem folgenden Klartextblock XOR verknüpft wird. Analog wird mit allen folgenden Blöcken verfahren.

Im Regelfall ist der Initialisierungsvektor mit null vorbelegt. Allerdings wird in manchen Systemen als Ersatz von temporären Schlüsseln eine sitzungsspezifische Zufallszahl in ihn geschrieben. Diese muss natürlich bei einer nachfolgenden Entschlüsselung bekannt sein.

Mehrfachverschlüsselung

Zusätzlich zu den vier Betriebsmodi eines Blockverschlüsselungsalgorithmus existiert noch eine weitere Variante, die zur Erhöhung der Sicherheit Verwendung findet. Sie wird aber praktisch nur beim DES aufgrund des kleinen Schlüsselraums eingesetzt. Prinzipiell lässt sie sich aber bei allen Blockverschlüsselungsverfahren, die keine Gruppe sind, einsetzen. Falls ein Verschlüsselungsalgorithmus diese Eigenschaft besitzt, würde eine Zweifachverschlüsselung mit zwei verschiedenen Schlüsseln keine Erhöhung der kryptografischen Sicherheit bedeuten, da das Ergebnis identisch der Verschlüsselung mit einem dritten Schlüssel wäre. Dies bedeutet, dass durch eine Zweifachverschlüsselung mit einem Algorithmus, der die Gruppeneigenschaft hat, der Schlüsselraum in seiner Größe nicht verändert wird, da ein Angreifer lediglich den dritten Schlüssel herausfinden müsste, um auf das Gleiche Ergebnis wie die vormalige Verschlüsselung mit zwei verschiedenen Schlüsseln zu kommen.

Schlüsseltext = enc (*Schlüssel 2; (enc (Schlüssel 1; Klartext))*)

Der DES ist aber keine Gruppe, also könnte man prinzipiell eine zweimalige Verschlüsselung mit zwei unterschiedlichen Schlüsseln durchführen, ohne dass es einen dritten Schlüssel gibt, mit dem man nach einer Einfachverschlüsselung das Gleiche Ergebnis erhält.

Von Ralph C. Merkle und Martin E. Hellman wurde aber 1981 eine Angriffsmethode veröffentlicht, die „meet-in-the-middle-attack" genannt wird [Merkle 81] und sehr erfolgreich gegen jede Zweifachverschlüsselung mit Blockverschlüsselungsalgorithmen eingesetzt werden kann. Die Voraussetzung dafür ist die Kenntnis von mehreren Klartext-Schlüsseltext-Paaren. Das Funktionsprinzip beruht darauf, dass man alle möglichen Verschlüsselungen des Klartextes mit dem ersten der beiden Schlüssel berechnet und anschließend mit jedem möglichen zweiten Schlüssel das bekannte Endergebnis, d. h. den Schlüsseltext, entschlüsselt. Nun vergleicht man die im ersten Schritt erstellte Liste mit den Ergebnissen aus dem zweiten Schritt. Sobald man eine Übereinstimmung feststellt, hat man mit einer gewissen Wahrscheinlichkeit beide Schlüssel gefunden. Zur Erhöhung der Sicherheit, dass man die richtigen Schlüssel gefunden hat, vergleicht man mit weiteren bekannten Klartext-Schlüsseltext-Paaren. Daraus lässt sich absehen, dass der Aufwand gegenüber einem üblichen Angriff, bei dem ebenfalls der gesamte Schlüsselraum durchsucht werden muss, nur unwesentlich größer ist. Deshalb verwendet man beim DES keine kaskadierte Zweifachverschlüsselung.

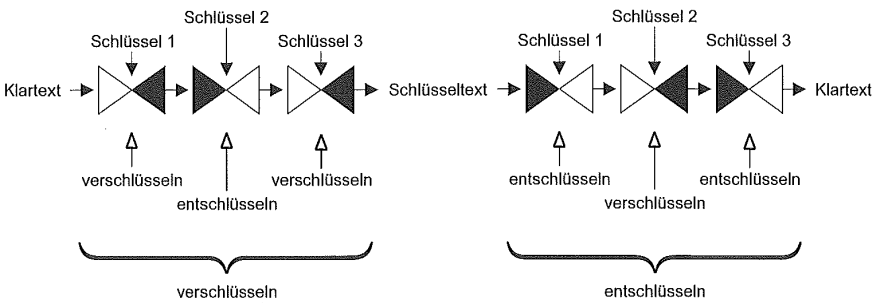


Bild 4.29 Der prinzipielle Ablauf der Verschlüsselung beim Triple-DES-Verfahren im äußeren CBC-Mode. Üblicherweise wird Schlüssel 1 identisch mit Schlüssel 3 gewählt, sodass die effektive Schlüssellänge 2 · 56 bit (= 112 Bit) beträgt. Seltener werden drei unabhängige Schlüssel mit einer daraus resultierenden Länge von 3 · 56 bit (= 168 Bit) benutzt.

Stattdessen benutzt man ein Verfahren, das Triple-DES genannt wird. Dazu schaltet man drei DES-Operationen im CBC-Mode mit abwechselnder Ver- und Entschlüsselung hintereinander. Die Entschlüsselung des so dreifach verschlüsselten Blocks erfolgt durch Umdrehung der Reihenfolge der Operationen, also Entschlüsselung – Verschlüsselung – Entschlüsselung. Sind alle drei Schlüssel identisch, so erhält man durch die abwechselnde Ver- und Entschlüsselung das Gleiche Ergebnis wie bei einer einfachen Verschlüsselung. Dies ist der Grund, warum nicht drei Verschlüsselungsoperationen hintereinander angeordnet sind.

Finden die DES-Operationen mit den drei Schlüsseln unmittelbar hintereinander auf einen Klartextblock Anwendung, dann spricht man von einem DES im inneren CBC-Mode (*inner CBC mode*). Wird der Klartextblock mit dem ersten Schlüssel komplett verschlüsselt und in dieser Form dann analog weiter verarbeitet, dann nennt man dies äußerer CBC-Mode (*outer CBC mode*). Der äußere CBC-Mode ist resistenter gegen Angriffe und wird deshalb im Allgemeinen empfohlen [Schneier 96].

Der Triple-DES hat auch noch mehrere andere Namen, wie DES-3, 3-DES und TDES. Triple-DES oder TDES bedeutet eigentlich nur, dass drei Schlüssel à 56 Bit verwendet werden. Falls der Erste und Dritte gleich sind, nennt man dies auch 2-DES, sind jedoch alle drei Schlüssel verschieden, dann benutzt man oft 3-DES als Bezeichnung dafür. Korrekterweise muss beim Triple-DES damit immer die Schlüssellänge angegeben werden, damit der Algorithmus eindeutig festgelegt ist.

Der Triple-DES ist wesentlich sicherer als eine aufeinander folgende Zweifachverschlüsselung mit verschiedenen Schlüsseln, da der Meet-in-the-middle-Angriff hier nicht greift. Es werden statt eines 56-Bit-Schlüssels also drei benötigt, von denen der Erste und der Dritte aber üblicherweise gleich sind. Dies führt dann zu einer Schlüssellänge von 2 · 56 Bit. Damit ist dieses Verfahren datenkompatibel zum normalen DES und erfordert außer dem doppelt so großen Schlüssel keinerlei Mehraufwand. Gerade dies ist ein Hauptargument, warum der Triple-DES in Chipkarten großen Einsatz findet. Aufgrund der deutlich höheren Sicherheit gegenüber einer einzelnen Verschlüsselung benutzt man es vor allem zur Schlüsselableitung oder zur Sicherung von sehr sensiblen Daten, wie z. B. der Übertragung von Schlüsseln.

Tabelle 4.13 Überblick der in Chipkarten verwendeten symmetrischen und asymmetrischen Kryptoalgorithmen mit den Eingangs- und Ausgangsparametern

Name	Typ	Länge Klartext	Länge Schlüsseltext	Länge Schlüssel
DES	symmetrisch	8 Byte	8 Byte	56 Bit
Triple-DES (mit 2 Schlüssel)	symmetrisch	8 Byte	8 Byte	2 · 56 Bit (= 112 Bit)
Triple-DES (mit 3 Schlüssel)	symmetrisch	8 Byte	8 Byte	3 · 56 Bit (= 168 Bit)
IDEA	symmetrisch	8 Byte	8 Byte	128 Bit
AES	symmetrisch	16 Byte	16 Byte	128 Bit (= 16 Byte) 192 Bit (= 24 Byte) 256 Bit (= 32 Byte)
RSA	asymmetrisch	512 Bit (= 64 Byte) 768 Bit (= 96 Byte) 1 024 Bit (= 128 Byte) 2 048 Bit (= 256 Byte) 4 096 Bit (= 512 Byte)	512 Bit (= 64 Byte) 768 Bit (= 96 Byte) 1 024 Bit (= 128 Byte) 2048 Bit (= 256 Byte) 4 096 Bit (= 512 Byte)	512 Bit (= 64 Byte) 768 Bit (= 96 Byte) 1 024 Bit (= 128 Byte) 2 048 Bit (= 256 Byte) 4 096 Bit (= 512 Byte)
DSS (512 Bit)	asymmetrisch	20 Byte	20 Byte	(64 + 20) Byte

4.7.2 Asymmetrische Kryptoalgorithmen

Im Jahr 1976 beschrieben Whitfield Diffie und Martin E. Hellman die Möglichkeit, einen Verschlüsselungsalgorithmus zu entwickeln, der auf zwei unterschiedlichen Schlüsseln basiert [Diffie 76]. Ein Schlüssel sollte dabei öffentlich sein, der andere geheim.

Dadurch wäre es möglich, dass jemand mit dem öffentlichen Schlüssel eine Nachricht verschlüsselt und nur der Besitzer des geheimen Schlüssels diese auch wieder entschlüsseln kann. Das Problem des Austausches und Verteilens von geheimen symmetrischen Schlüsseln wäre damit beseitigt und weitere Verfahren, wie etwa die digitale Signatur, die von jedem verifiziert werden kann, wären erstmals möglich.

Bild 4.30 Die Ver- und Entschlüsselung mit einem Public-Key-Algorithmus.

Der RSA-Algorithmus

Zwei Jahre später stellten Ronald L. Rivest, Adi Shamir und Leonard Adleman einen Algorithmus vor, der die obigen Voraussetzungen erfüllte [Rivest 78]. Der nach seinen drei Erfindern benannte RSA-Algorithmus ist der bekannteste und der am vielseitigsten einsetzbare asymmetrische Kryptoalgorithmus, der zurzeit verwendet wird. Das sehr einfache Funk-

tionsprinzip basiert auf der Arithmetik großer Ganzzahlen. Die beiden Schlüssel werden auf der Grundlage von zwei großen Primzahlen erzeugt.[6]

Die Ver- und Entschlüsselung lässt sich mathematisch folgendermaßen ausdrücken:

$$\text{Verschlüsseln} : y = x^e \bmod n$$
$$\text{Entschlüsseln} : x = y^e \bmod n$$
$$\text{mit} : n = p \cdot q$$

Bild 4.31 Die Ver- und Entschlüsselung mit dem RSA-Algorithmus.

x:	Klartext	e:	öffentlicher Schlüssel
y:	Schlüsseltext	d:	geheimer Schlüssel
		n:	öffentlicher Modulus
		p, q:	geheime Primzahlen

Vor der Verschlüsselung muss der Klartextblock auf die entsprechende Blocklänge aufgefüllt werden. Dies kann beim RSA-Algorithmus je nach verwendeter Schlüssellänge unterschiedlich sein. Die Verschlüsselung selber geschieht dabei durch Potenzieren des Klartextes mit einer anschließenden Modulo-Operation. Das Ergebnis dieser Berechnung ist der Schlüsseltext. Dieser kann nur mehr bei Kenntnis des geheimen Schlüssels entschlüsselt werden, welche analog der Verschlüsselung erfolgt.

Die Sicherheit des Algorithmus basiert also damit auf dem Faktorisierungsproblem großer Zahlen. Es ist sehr einfach, den öffentlichen Modulus aus den beiden Primzahlen durch Multiplikation zu berechnen, aber sehr schwierig, den Modulus wieder in seine beiden Primfaktoren zu zerlegen, da es keine effektiven Algorithmen dafür gibt.

Für die Exponentation der großen Zahlen im Rahmen von Ver- und Entschlüsselung würde das RAM einer Chipkarte nicht ausreichen, da die Zahlen vor der Modulo-Berechnung sehr groß werden. Deshalb verwendet man hier die so genannte modulare Exponentation, bei der die Zwischenergebnisse bei der Berechnung nie größer werden als der Modulowert. Muss man beispielsweise x^2 mod n berechnen, so wird man nicht $(x \cdot x)$ mod n rechnen, da das Zwischenergebnis (d. h. $x \cdot x$) vor der Reduktion der Zahlenlänge durch die Modulo-Berechnung eine unnötig große Zahl wäre. Stattdessen wird man $((x \bmod n) \cdot (x \bmod n) \bmod n)$ berechnen, was mathematisch kein Unterschied zum Ausgangswert ist. Der Vorteil ist, dass dies wesentlich weniger Rechenschritte und Speicherplatz erfordert, da die Zwischenergebnisse sofort in ihrer Länge reduziert werden.

Ein weiteres Verfahren, um den RSA-Algorithmus zu beschleunigen, besteht darin, den chinesischen Restklassensatz (*chinese remainder theorem*) für die Berechnung zu benutzen.[7] Die Verwendung des chinesischen Restklassensatzes setzt allerdings voraus, dass

[6] Whitfield Diffie und Martin E. Hellman waren die Erfinder des Prinzips der Public-Key-Algorithmen und Ronald L. Rivest, Adi Shamir und Leonard Adleman der ersten Realisierung. Allerdings waren Public-Key-Algorithmen im Bereich der militärischen Kryptografie bereits einige Jahre vorher entdeckt worden. Die Mitarbeiter des britischen Government Communication Headquarters (GCHQ) James Ellis, Clifford Cocks und Malcom Williamson entwickelten in den Jahren 1969 bis 1973, wenige Jahre vor den zivilen Erfindern, sowohl das Prinzip von Public-Key-Algorithmen als auch das heute als RSA-Algorithmus bekannte asymmetrische Verschlüsselungsverfahren. Leider waren diese Kryptologen zu Stillschweigen verpflichtet und durften bis Ende des Jahres 1997 über ihre Arbeiten nichts veröffentlichen [Levy 99].

[7] Nähere Informationen zu beiden Verfahren: [Simmons 92, Schneier 96].

man die beiden geheimen Primzahlen p und q besitzt, weshalb das Verfahren nur bei der Entschlüsselung (d. h. das Signieren) benutzt werden kann.

Der geheime Schlüssel sollte so groß wie möglich sein, damit Angriffe erschwert werden. Öffentlicher und geheimer Schlüssel können unterschiedliche Längen haben, was auch üblicherweise der Fall ist, da es die Rechenzeit zur Prüfung einer digitalen Signatur erheblich reduziert, wenn der öffentliche Schlüssel möglichst klein ist. Als öffentlicher Schlüssel wird oft die vierte Fermatsche Zahl verwendet. Diese Primzahl hat den Wert $(2^{16}) + 1 = 65537$ und eignet sich aufgrund ihrer Kürze sehr gut für schnelle Überprüfungen von digitalen Signaturen. Die Zahlen 3 und 17 finden in diesem Zusammenhang ebenfalls Verwendung.

Tabelle 4.14 Typische öffentliche Schlüssel für den RSA-Algorithmus

öffentlicher Schlüssel	Bemerkung
$2 = °10°$	einzige gerade Primzahl; wird für das Rabin-Verfahren verwendet
$3 = (2^1) + 1 = °11°$	kleinste ungerade Primzahl
$17 = (2^4) + 1 = °1\ 0001°$	---
$65\,537 = (2^{16}) + 1 = °1\ 0000\ 0000\ 0000\ 0001°$	4te Fermatsche Zahl F_4

Gelänge es nun einem Angreifer, den öffentlichen Modulo in seine beiden Primfaktoren aufzuspalten, so könnte er die gesamte Schlüsselberechnung nachvollziehen. Dies ist bei einem kleinen Wert, wie im Beispiel der Zahl 33, sehr einfach möglich, doch bei großen Zahlen existiert dafür bis heute kein schneller Algorithmus. Können die beiden Primfaktoren gefunden werden, so ist damit das System gebrochen, da der geheime Schlüssel bekannt ist.[8]

Vorangehend wurde für RSA-Schlüssel gefordert, dass sie hinreichend groß sind. Zurzeit betrachtet man 512 Bit (= 64 Byte) lange Schlüssel als die untere Grenze. Allerdings finden auch schon 768 Bit (= 96 Byte) und 1024 Bit (= 128 Byte) lange Schlüssel Verwendung. In den nächsten Jahren werden bald die ersten 2048 Bit (= 256 Byte) Schlüssel zum Einsatz kommen. Mit steigender Schlüssellänge steigt auch der Rechenaufwand zur Ver- und Entschlüsselung. Diese Steigerung ist jedoch nicht linear, sondern annähernd exponentiell.

Auf Chipkarten-Mikrocontrollern mit ihren 8 Bit breiten CPUs besteht normalerweise keine Möglichkeit, eine RSA-Berechnung innerhalb einiger Minuten Rechenzeit durchzuführen. Doch existieren mittlerweile Mikrocontroller mit zusätzlichen Rechenwerken, die besonders auf schnelle Exponentation hin entwickelt wurden. Mit diesen ist es möglich, RSA-Berechnungen in akzeptabler Zeit und mit vertretbarem Softwareaufwand durchzuführen. Der Codeumfang für den beschriebenen Hardware-unterstützten 512-Bit RSA-Algorithmus bewegt sich im Bereich um 300 Bytes. Bei 768 Bit und 1024 Bit Schlüssellänge benötigt man ungefähr 1 kByte Assemblercode in der Chipkarte.

[8] Bereits im Sommer 1994 gelang es, mit ca. 1 600 über Internet zusammengeschlossenen Computern in 8 Monaten einen 426-Bit-Schlüssel zu brechen. Die gesamte Rechenleistung betrug dabei 5 000 MIPS Jahre. Ein 100 MHz Pentium PC hat dazu im Vergleich 50 MIPS. Im Jahr 1999 gelang dann die Faktorisierung eines 512 Bit Schlüssels in 7 Monaten mit 292 vernetzten Computern.

Betrachtet man die Tabelle 4.9, so sieht man, dass die Anzahl der möglichen Primzahlen selbst für 512-Bit-Schlüssel eine solche Größe aufweist, dass es nie zu Interferenzen zwischen zwei verschiedenen Schlüsselpaaren kommen kann.[9]

Tabelle 4.15 Typische RSA-Schlüssellängen mit charakteristischen Werten. Der Quotient ZR/PZ gibt die Anzahl der Nicht-Primzahlen zu den Primzahlen an. Der reziproke Wert davon ist die Wahrscheinlichkeit, dass eine Zufallszahl dieses Zahlenraums prim ist. Für die Zeitdauer einer RSA-Schlüsselgenerierung ist das von großer Bedeutung.

Schlüssellänge	**maximale Zahl der Dezimalstellen**	**Größe des Zahlenraums (= ZR)**	**Anzahl der Primzahlen im Zahlenraum (= PZ)**	**ZR/PZ**
8 Bit = 1 Byte	3	256	54	≈ 4,7
40 Bit = 5 Byte	13	$\approx 1{,}1 \cdot 10^{12}$	$\approx 3{,}9 \cdot 10^{10}$	≈ 28
512 Bit = 64 Byte	155	$\approx 1{,}3 \cdot 10^{154}$	$\approx 3{,}8 \cdot 10^{151}$	≈ 342
768 Bit = 96 Byte	232	$\approx 1{,}6 \cdot 10^{231}$	$\approx 2{,}9 \cdot 10^{228}$	≈ 552
1024 Bit = 128 Byte	309	$\approx 1{,}8 \cdot 10^{308}$	$\approx 2{,}5 \cdot 10^{305}$	≈ 720
2048 Bit = 256 Byte	617	$\approx 3{,}2 \cdot 10^{616}$	$\approx 2{,}3 \cdot 10^{613}$	≈ 1391
4096 Bit = 512 Byte	1234	$\approx 1{,}0 \cdot 10^{1233}$	$\approx 2{,}5 \cdot 10^{1229}$	≈ 4000

Es ist aber eine der Stärken des RSA-Algorithmus, dass er nicht auf eine bestimmte Schlüssellänge fixiert ist, wie dies z. B. beim DES der Fall ist. Benötigt man mehr Sicherheit, so kann man, ohne den Algorithmus zu ändern, größere Schlüssellängen verwenden. Der RSA ist damit skalierbar. Allerdings muss man dabei immer die Rechenzeit und den Speicherbedarf im Auge behalten, da selbst 768-Bit-Schlüssel momentan noch als sicher gelten. Bei den heutigen Algorithmen zur Faktorisierung kann man als Faustwert annehmen, dass eine Erhöhung der Schlüssellänge um 15 Bit zu einer Verdopplung des Faktorisierungsaufwands führt.[10] Einen sehr guten Überblick über weltweit vorhandene und notwendige Rechenleistungen zur Faktorisierung und auch zum Brechen von symmetrischen Kryptoalgorithmen findet sich bei Andrew Odlyzko [Odlyzko 95].

Die Verwendung des RSA zur Verschlüsselung von Daten wird wegen der langen Rechenzeiten selten genutzt, obwohl sie sehr sicher ist. Das Haupteinsatzgebiet ist im Bereich der digitalen Signatur, da hier die Vorteile dieses asymmetrischen Verfahrens voll zur Geltung kommen.

---

[9] Die größte Zahl, die man mit 512 Bit darstellen kann, ist:
$(2^{512})-1$ = 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

[10] Die im Januar 1998 größte bekannte Primzahl hat 909 256 Stellen und den Wert $2^{3\,402\,377}$-1. Eine ausführliche und aktuelle Aufstellung der größten Primzahlen mit viel Hintergrundinformationen findet man unter [UTM].

Der größte Nachteil des RSA-Algorithmus bei der Verwendung in Chipkarten ist der benötigte Speicherplatz für die Schlüssel. Die aufwendige Schlüsselgenerierung ist in bestimmten Fällen ebenfalls problematisch.

Die Verbreitung von RSA hemmt die in einigen Ländern gemachten Patentansprüche zum Algorithmus sowie die großen Restriktionen bei der Ein- und Ausfuhr von den RSA benutzenden Geräten. Chipkarten mit einem Koprozessor für den RSA-Algorithmus fallen unter diese Bestimmungen, wobei sich die internationalen Vertriebsmöglichkeiten erschweren.

Tabelle 4.16 Beispiele für Berechnungszeiten bei RSA-Ver- und Entschlüsselung in Abhängigkeit von der Schlüssellänge. Die angegebenen Werte können zum Teil erheblich differieren, da sie stark vom verwendeten Mikrocontroller, der Bitstruktur des Schlüssels und der Verwendung des chinesischen Restklassensatzes (nur beim Signieren) abhängen.

Realisierung	Modus	512 Bit	768 Bit	1024 Bit	2048 Bit
Chipkarte ohne NPU, 8 Bit CPU, 3,5 MHz Takt	Signieren	20 min	---	---	---
Chipkarte ohne NPU, 8 Bit CPU, 3,5 MHz Takt (mit chinesischem Restklassensatz)	Signieren	6 min	---	---	---
Chipkarte mit NPU und 3,5 MHz	Signieren	308 ms	910 ms	2000 ms	---
Chipkarte mit NPU und 3,5 MHz (mit chinesischem Restklassensatz)	Signieren	84 ms	259 ms	560 ms	---
Chipkarte mit NPU und 4,9 MHz	Signieren	220 ms	650 ms	1400 ms 65 ms	---
Chipkarte mit NPU und 3,5 MHz	Verifizieren			1,04 s	---
Chipkarte mit NPU und 4,9 MHz (mit chinesischem Restklassensatz)	Signieren	60 ms	185 ms	400 ms	---
Chipkarte mit NPU und PLL (mit chinesischem Restklassensatz)	Signieren	---	---	---	40 ms
Chipkarte mit NPU und PLL	Verifizieren	---	---	---	840 ms
PC (Pentium, 200 MHz)	Signieren	12 ms	46 ms	60 ms	390 ms
PC (Pentium, 200 MHz)	Verifizieren	2 ms	4 ms	6 ms	20 ms
RSA-Hardwarebaustein	Signieren	1,6 ms	---	---	---

Schlüsselgenerierung für RSA

Die Erzeugung von Schlüsseln für den RSA-Algorithmus erfolgt nach einem einfachen Schema, anhand dessen hier ein kleines Beispiel durchgerechnet werden soll:

1. Zuerst werden zwei Primzahlen p und q gesucht. $p = 3, q = 11$
2. Nun berechnet man den öffentlichen Modulo. $n = p \cdot q = 33$
3. Berechnung einer Hilfsvariablen z für die Schlüsselerzeugung. $z = (p - 1) \cdot (q - 1)$
4. Berechnung des öffentlichen Schlüssels e mit den folgenden Eigenschaften: $(e < z)$ und $(ggt(z, e) = 1)$, d. h. größter gemeinsamer Teiler von z und e ist 1. Da es mehrere Zahlen mit dieser Eigenschaft gibt, wähle man eine aus. $e = 7$
5. Berechnung des geheimen Schlüssels d mit der Eigenschaft: $(d \cdot e) \bmod z = 1$. $d = 3$

Nun ist die Schlüsselberechnung abgeschlossen. Der öffentliche und der geheime Schlüssel können nun an einem weiteren Zahlenbeispiel zur Ver- und Entschlüsselung mit dem RSA-Algorithmus getestet werden.

1.	Als Klartext x (mit x < n) verwendet man die Zahl 4.	$x = 4$
2.	Verschlüsselung	$y = 4^7 \bmod 33 = 16$
3.	Es ergibt sich aus der Berechnung ein Schlüsseltext y mit dem Wert 16.	$y = 16$
4.	Entschlüsselung	$x = 16^3 \bmod 33 = 4$

Das Ergebnis der Entschlüsselung des Schlüsseltextes ist, wie erwartet, wiederum der Klartext.

In der Praxis stellt sich die Schlüsselgenerierung allerdings etwas aufwendiger dar, da es ziemlich schwierig ist, große Zahlen daraufhin zu testen, ob sie prim sind. Man kann dazu nicht mehr das altbekannte Sieb des Eratosthenes benutzen, weil für diesen Algorithmus alle Primzahlen, die kleiner als die zu prüfende Zahl sind, bekannt sein müssen. Dies ist bei Zahlen im Bereich von 512 Bit Länge praktisch unmöglich.

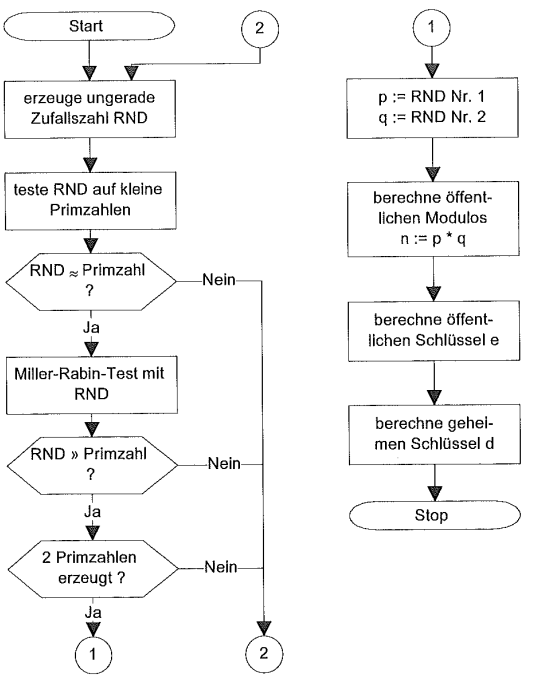


Bild 4.32 Grundsätzliche Vorgehensweise der Generierung von Schlüssel für den RSA-Algorithmus in Chipkarten.

Deshalb wendet man probabilistische Primzahlentests an, bei denen man eine Wahrscheinlichkeitsaussage erhält, ob die zu prüfende Zahl eine Primzahl ist. Typische Vertreter dieser Tests sind der Miller-Rabin-Test und der Solovay-Strassen-Test.[11] Um diese zeit-

[11] Eine Beschreibung der Verfahren und Algorithmen findet sich bei Alfred Menezes [Menezes 97].

aufwendigen Tests nicht unnötig oft zu benutzen, prüft man die per Zufall erzeugten Zahlen vorab auf kleine Primfaktoren. Ist die generierte Zufallszahl durch eine kleine Primzahl, wie 2, 3, 5, 7, ohne Rest teilbar, dann kann sie selber nicht mehr prim sein, da sie offensichtlich aus mehreren Primfaktoren zusammengesetzt ist. Steht fest, dass die zu prüfende Zahl keine kleinen Primfaktoren besitzt, prüft man erst dann mit einem Primzahlentest wie dem Miller-Rabin-Test. Das Prinzip dazu ist in Bild 4.32 veranschaulicht und im Anhang der IEEE 1363 im Detail beschrieben.[12]

Die Algorithmen für die RSA-Schlüsselgenerierung haben eine Besonderheit. Die Zeit für die Erzeugung eines Schlüsselpaares (d. h. geheimer und öffentlicher Schlüssel) ist nur statistisch vorhersagbar. Es kann also nur angegeben werden, dass mit einer bestimmten Wahrscheinlichkeit die Schlüsselgenerierung eine bestimmte Zeit benötigt. Eine fixe Aussage im Sinne von „. . . benötigt x Sekunden" ist durch die notwendigen und nicht deterministisch vorhersagbaren Primtests der Zufallszahlen nicht möglich.

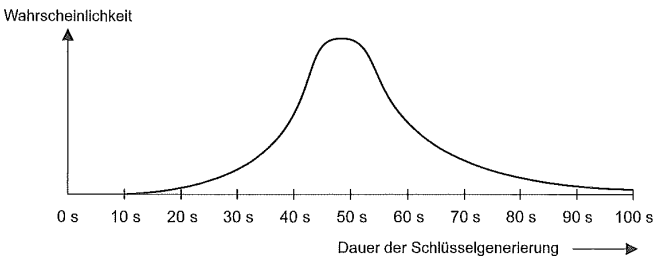


Bild 4.33 Das typische Zeitverhalten eines probabilistischen Algorithmus zur Erzeugung von Schlüsselpaaren für den RSA-Algorithmus. Die Ordinate zeigt die Wahrscheinlichkeiten, dass für eine 1 024-Bit-Schlüsselgenerierung in einer Chipkarte eine bestimmte Zeitdauer benötigt wird. Im Beispiel sind dafür durchschnittlich 50 s notwendig. Die Fläche unter der Kurve hat die Wahrscheinlichkeit 1.

Tabelle 4.17 Tabelle mit Beispielen für den Zeitbedarf der Generierung eines öffentlichen und geheimen Schlüssels für den asymmetrischen Kryptoalgorithmus RSA. Eine exakte Zeitdauer kann nicht angegeben werden, da die Dauer der Schlüsselgenerierung unter anderem davon abhängt, ob die erzeugten Zufallszahlen prim sind.

Generierung eines Schlüsselpaares (öffentlich, geheim) für den RSA-Algorithmus	**typischer Zeitbedarf**	**möglicher Zeitbedarf**
Chipkarte, 512 Bit Schlüssellänge, 3,5 MHz	6 s	≈ 1 s . . . ≈ 20 s
Chipkarte, 1 024 Bit Schlüssellänge, 3,5 MHz	14 s	≈ 6 s . . . ≈ 40 s
Chipkarte, 2 048 Bit Schlüssellänge, 18 MHz	80 s	---
PC (Pentium, 200 MHz), 512 Bit Schlüssellänge	0,5 s	---
PC (Pentium, 200 MHz), 1 024 Bit Schlüssellänge	2 s	---
PC (Pentium, 200 MHz), 2 048 Bit Schlüssellänge	36 s	---

[12] Viele Hinweise und zu beachtende Kriterien über die Generierung von Primzahlen für den RSA-Algorithmus finden sich in dem Artikel von Robert Silverman [Silverman 97].

Der DSS-Algorithmus

Mitte des Jahres 1991 veröffentlichte das NIST (US National Institute of Standards and Technology) einen Entwurf für einen kryptografischen Algorithmus zum Signieren von Nachrichten. Der mittlerweile in einer US Norm (FIPS 186) standardisierte Algorithmus hat die Bezeichnung DSA (*digital signature algorithm*) und die ihn beschreibende Norm den Namen DSS (*digital signature standard*). Der DSA ist neben dem RSA-Algorithmus das zweite im großen Rahmen eingesetzte Verfahren zur Erstellung von digitalen Signaturen. Der DSA ist eine Modifikation des ElGamal-Verfahrens.

Hintergrund für die Normierung dieses Signaturverfahrens war die Suche nach einem Verfahren, mit dem nur Signaturen erstellt werden können und keine Verschlüsselung von Daten möglich ist. Durch diese Forderung ist der DSS-Algorithmus aufwendiger als der RSA-Algorithmus. Es ist jedoch eine Möglichkeit gefunden worden, auch mit diesem Algorithmus Daten zu verschlüsseln [Simmons 93].

Im Gegensatz zum RSA-Algorithmus beruht die Sicherheit des DSS nicht auf dem Faktorisierungsproblem großer Zahlen, sondern auf dem diskreten Logarithmusproblem. Die Berechnung von $y = a^x \bmod p$ ist auch für große Zahlen in kurzer Zeit möglich. Umgekehrt erfordert es aber sehr viel Rechenaufwand, den Wert für x bei gegebenen y, a und p zu bestimmen.

Es ist bei allen Signieralgorithmen erforderlich, die zu unterschreibende Nachricht mit einem Hash-Algorithmus auf eine feste Länge zu reduzieren. Für den DSS wurde deshalb vom NIST ein dazu passender Algorithmus veröffentlicht. Dieser trägt die Bezeichnung SHA-1 (*secure hash algorithm*).[13] Er erzeugt einen 160 Bit langen Hashwert über eine Nachricht beliebiger Länge und ist eine Abwandlung des MD5 Hash-Algorithmus.

Beim DSS-Algorithmus wird im Übrigen genauso wie beim RSA-Algorithmus nur mit ganzen Zahlen gerechnet. Um nun mit dem DSA eine digitale Signatur zu berechnen, müssen vorab die folgenden globalen Werte ermittelt werden:

p	öffentlich	Primzahl, 512 bis 1 024 Bit lang, die Länge muss geradzahlig durch 64 teilbar sein
q	öffentlich	Primfaktor von $p - 1$, 160 Bit lang
g	öffentlich	$g = h^{((p-1)/q)}$ mit h als einer ganzen Zahl, für die gilt: $h < p - 1$ und $g > 1$

Der geheime Schlüssel x muss das folgende Kriterium erfüllen:

x	geheimer Schlüssel	$x < q$

Der öffentliche Schlüssel y wird folgendermaßen berechnet:

y	öffentlicher Schlüssel	$y = g^x \bmod p$

Sind alle notwendigen Schlüssel und Zahlen ermittelt, kann eine Nachricht m unterschrieben werden:

1.	generiere Zufallszahl k mit $k < q$	
2.	Berechnung: Hashwert von m	H(m)
3.	Berechnung	$r = (g^k \bmod p) \bmod q$
4.	Berechnung	$s = k^{-1} (H(m) + x \cdot r)) \bmod q$

[13] Siehe auch Abschnitt 4.9 „Hash-Funktionen und FIPS 180-1".

Die beiden Werte r und s sind die digitale Signatur der Nachricht. Sie besteht beim DSA aus zwei Zahlen und nicht aus einer wie beim RSA-Algorithmus.
Die Prüfung der Signatur wird in den folgenden Zeilen durchgeführt:

1. Berechnung $w = s^{-1} \bmod q$
2. Berechnung $u1 = (H(m) \cdot w) \bmod q$
3. Berechnung $u2 = (r \cdot w) \bmod q$
4. Berechnung $v = ((g^{u1} \cdot y^{u2}) \bmod p) \bmod q$

Falls nun die Bedingung $v = s$ erfüllt ist, wurde die Nachricht m nicht verändert, und die digitale Signatur ist echt.

Tabelle 4.18 Beispiele für Berechnungszeiten für den DSA, getrennt in Ver- und Entschlüsselung in Abhängigkeit vom Takt. Die angegebenen Werte können zum Teil erheblich differieren, da sie stark von der Bitstruktur des Schlüssels abhängen. Die Geschwindigkeit kann durch Vorberechnung erhöht werden.

Realisierung	Überprüfung einer 512-Bit-Signatur	Erstellung einer 512-Bit-Signatur
Chipkarte mit 3,5 MHz Takt	130 ms	70 ms
Chipkarte mit 4,9 MHz Takt	90 ms	50 ms
PC (80 386, 33 MHz)	16 s	35 ms

Der RSA-Algorithmus hat in der Praxis gegenüber dem DSS-Algorithmus, der bislang nur sehr verhalten eingesetzt wird, eine größere Verbreitung gewonnen. Die ursprüngliche Absicht, mit dem DSS einen Signieralgorithmus zu normieren, mit dem man nicht verschlüsseln kann, ist weitgehend fehlgeschlagen. Auch ist die Komplexität des Verfahrens der Verbreitung nicht förderlich. Jedoch ist die bestehende Norm und der damit verbundene politische Druck, digitale Signaturen in Verbindung mit DSS und SHS zu erstellen, für viele Institutionen ein starkes Argument.

Elliptische Kurven als asymmetrische Kryptoalgorithmen

Neben den beiden bekannten asymmetrischen kryptografischen Algorithmen RSA und DSA gibt es im Umfeld von Chipkarten eine dritte Spielart der Kryptografie, welche für digitale Signatur und Schlüsselaustausch benutzt wird – die elliptischen Kurven (*elliptic curves – EC*).

Im Jahr 1985 schlugen Victor Miller und Neal Koblitz unabhängig voneinander vor, elliptische Kurven für die Konstruktion von asymmetrischen Kryptoalgorithmen zu verwenden. Die Eigenschaften von elliptischen Kurven sind dafür gut geeignet, und im Laufe der folgenden Jahre wurden aus dem Vorschlag in der Praxis einsetzbare Kryptosysteme, welche meist allgemein ECC (*elliptic curve cryptosystem*) genannt werden.

Elliptische Kurven sind zusammenhängende ebene Kurven mit der Gleichung $y^2 = x^3 + ax + b$ in einem endlichen Körper. Kein Punkt der Kurve darf ein singulärer Punkt sein, d. h. es gilt $4a^2 + 27b^2 \neq 0$. Im Bereich der Kryptografie werden die endlichen Körper GF(p), GF(2^n) und GF(p^n) verwendet, mit p als Primzahl und n als positiver Ganzzahl mit $n \geq 1$.

Die zu Kryptosystemen auf der Grundlage von elliptischen Kurven gehörige Mathematik ist verhältnismäßig schwierig, weshalb hier auf das Buch von Alfred Menezes [Menezes 93] verwiesen wird. Einen guten Überblick über elliptische Kurven und andere asymmetrische Kryptoverfahren gibt auch die sehr ausführliche Norm IEEE 1363 über Public Key Cryptography und die Normenreihe ISO/IEC 15946 über elliptische Kurven.

Der große Vorteil von Kryptosystemen auf der Grundlage von elliptischen Kurven besteht darin, dass diese eine weitaus geringere Rechenleistung als beispielsweise RSA benötigen und dass die Schlüssellängen bei gleicher kryptografischer Stärke wesentlich kleiner sind. So benötigt man zum Brechen eines ECC mit Schlüssellänge 160 Bit in etwa die gleiche Rechenleistung wie für den RSA-Algorithmus mit 1024 Bit. ECCs mit 256 Bit entsprechen innerhalb dieses Vergleichskriteriums dem RSA mit 2048 Bit und 320 Bit EC-Schlüssel entsprechen ungefähr 5120 Bit RSA-Schlüssel. Die kryptografische Stärke und die verhältnismäßig kurze Schlüssellänge sind auch die beiden Gründe, warum ECCs gerade im Umfeld von Chipkarten zu finden sind.

Die heutigen numerischen Recheneinheiten moderner Chipkarten-Mikrocontroller unterstützen im Regelfall ECCs, sodass eine verhältnismäßig hohe Rechengeschwindigkeit zur Verfügung steht. Analog dem RSA-Algorithmus ist die Schlüssellänge eine wichtige Kennzeichnung eines asymmetrischen Kryptoalgorithmus.

Interessanterweise benötigen Kryptoalgorithmen auf der Grundlage elliptischer Kurven eine so geringe Rechenleistung, dass sie sogar in Mikrocontrollern ohne Koprozessor implementiert werden können. Typische Zeiten für die Erstellung und Überprüfung von Signaturen befinden sich in Tabelle 4.18.

Die Erzeugung eines 160-Bit-ECC-Schlüsselpaares bei 3,5 MHz Takt auf einem 8-Bit-Mikrocontroller mit einer ca. 10 kByte großen Tabelle benötigt ohne Koprozessor etwa 1 s und kann durch einen Koprozessor auf 200 ms reduziert werden.

Tabelle 4.19 Beispiele für Berechnungszeiten eines Kryptoalgorithmus, der auf elliptischen Kurven in GF(p) beruht. Die überdurchschnittlich guten Zeitwerte für eine Chipkarte ohne Koprozessor wurden durch den Einsatz einer ca. 10 kByte großen Tabelle erreicht, sodass bestimmte zeitaufwendige Rechenschritte durch Zugriff auf die Tabelle beschleunigt werden können.

Realisierung	**Erstellung einer 160-Bit-Signatur**	**Überprüfung einer 160-Bit-Signatur**
Chipkarte mit 3,5 MHz Takt und 8-Bit-Prozessor	1 s	4 s
Chipkarte mit 3,5 MHz Takt und numerischem Koprozessor	150 ms	450 ms
PC (Pentium III, 500 MHz)	10 ms	20 ms

Gegen die Verwendung von elliptischen Kurven im Bereich von asymmetrischen Kryptoalgorithmen spricht, dass sie, obwohl schon seit langer Zeit bekannt, in der Kryptografie eine verhältnismäßig neue Entdeckung sind. Es wird wohl noch einige Zeit vergehen müssen, bis der Einsatz von ECCs in der umsichtigen Welt von Kryptologen und Anwendungs-Designern für Chipkarten alltäglich wird. Elliptische Kurven als Kryptosysteme haben jedoch gegenüber allen anderen asymmetrischen Verfahren den Vorteil, dass sie zurzeit die größte Sicherheit pro Bit Schlüssellänge bieten.

4.7.3 Padding

Im Chipkartenbereich setzt man den DES weitgehend in den beiden blockorientierten Betriebsarten ECB und CBC ein. Da jedoch nicht alle Daten in der Kommunikation mit einer Chipkarte ein Vielfaches der Blocklänge aufweisen, müssen die Blöcke u. U. aufgefüllt werden. Dieses Auffüllen eines Datenblockes auf ein Vielfaches der Blocklänge bezeichnet man als Padding.

Für den Empfänger eines solchen verschlüsselten Blockes ergibt sich nach der Entschlüsselung ein Problem. Er weiß nicht, wo die eigentlichen Daten aufhören und die Paddingbytes anfangen. Eine Lösungsmöglichkeit wäre, am Anfang der Nachricht die Länge anzugeben, doch würde dies die Struktur der Nachricht verändern, was in der Regel unerwünscht ist. Dies wäre besonders bei Daten aufwendig, die nicht immer verschlüsselt werden müssen, da man in diesem Falle dann kein Padding benötigt und damit auch keine Längenangabe. In vielen Fällen darf deshalb die Struktur der Nachricht nicht verändert werden.

Man muss also zur Kennzeichnung der Paddingbytes ein anderes Verfahren benutzen. Als typisches Beispiel sei hier der Algorithmus nach ISO/IEC 9797 im Detail aufgeführt, obwohl es noch eine Reihe weiterer Verfahren gibt: Im ersten Paddingbyte nach den Nutzdaten ist das höchstwertige Bit (msb) auf eins gesetzt. Damit hat dieses Byte den hexadezimalen Wert '80'. Falls noch weitere Paddingbytes notwendig sind, haben diese den Wert '00'. Der Empfänger dieser gepaddeten Nachricht sucht nun vom Ende zum Anfang nach einem gesetzten Bit, oder nach dem Wert '80'. Hat er ihn gefunden, so weiß er, dass dieses Byte und alle nachfolgenden zum Padding-Bereich gehören und nicht Teil der Nachricht sind.

Wichtig ist in diesem Zusammenhang, dass der Empfänger weiß, ob in jedem Fall gepaddet wird oder nur, wenn es notwendig ist. Findet das Padding nur statt, wenn die Länge der zu verschlüsselnden Daten ganzzahlig durch die Blocklänge teilbar ist, muss dies der Empfänger auch berücksichtigen. Deshalb vereinbart man oft implizit, dass ein Padding immer stattfindet, was natürlich zu dem Nachteil führt, dass im ungünstigsten Fall ein Block mehr verschlüsselt, übertragen und entschlüsselt werden muss, als eigentlich notwendig ist.

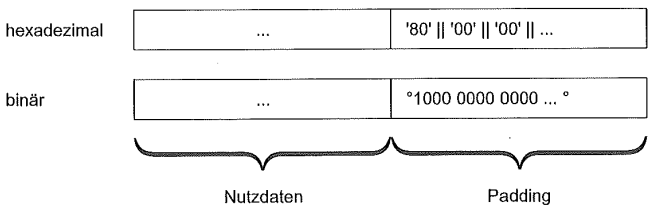


Bild 4.34 Das Padding von Daten nach ISO/IEC 9797, Methode 2.

In manchen Anwendungsfällen wird Padding nur mit dem Wert '00' durchgeführt. Der Grund dafür liegt darin, dass bei MAC-Berechnungen in der Regel mit '00' gepaddet wird. Verwendet man nun ein einheitliches Paddingverfahren, führt dies zur Ersparnis von Programmcode. Die Anwendung muss dann allerdings die genaue Struktur der Daten kennen, um so zwischen Nutzdaten und Paddingdaten zu unterscheiden.

Tabelle 4.20 Tabelle mit im Umfeld von Chipkarten typischen Padding-Formaten. Die zu paddenden Daten werden in der Tabelle als „Daten" bezeichnet.

Padding Format	Beschreibung
ISO/IEC 9797	Diese Padding-Formate werden für MACing bzw. Verschlüsselung verwendet. Methode 1: Die zu paddenden Daten werden mit '00' aufgefüllt. Formale Darstellung: Daten || n · '00' Methode 2: Den zu paddenden Daten wird '80' nachgestellt und anschließend wird mit '00' aufgefüllt. Formale Darstellung: Daten || '80' || n · '00'
ISO/IEC 9796-2	Diese Padding-Formate werden für digitale Signaturen verwendet. Den zu paddenden Daten wird dabei eine Bitsequenz vorangestellt, die mit °11° beginnt und mit °1° endet und dazwischen die zum Padding benötigte Anzahl °0° enthält. Den Daten wird zusätzlich das Kennzeichen 'BC' nachgestellt. Ergänzend kann in die Padding-Sequenz noch eine Zufallszahl integriert werden, sodass die zu paddenden Daten individualisiert werden. Formale Darstellung im Falle von byteweisem Padding: '60' || n · '00' || '01' || Daten || 'BC' Formale Darstellung im Falle von byteweisem Padding und individualisierten Daten: '60' || n · '00' || '01' || RND || Daten || 'BC'
PKCS #1	Diese Padding-Formate werden im Falle von Typ 1 für digitale Signaturen verwendet, das Typ-2-Format hingegen für MACing bzw. Verschlüsselung. Den zu paddenden Daten wird dabei ein Kennzeichen sowie ein statischer Wert oder eine Zufallszahl mit dem Padding entsprechender notwendiger Länge vorangestellt. Formale Darstellung von Typ 1: '00' || '01' || n · 'FF' || '00' || Daten Formale Darstellung von Typ 2: '00' || '02' || n · RND || '00' || Daten

4.7.4 Message Authentication Code/Cryptographic Checksum

Vielfach wichtiger als die Geheimhaltung ist die Authentizität einer Nachricht. Der Begriff „Authentizität" besagt, dass eine Nachricht unverändert und nicht manipuliert, also authentisch ist. Dazu stellt man der eigentlichen Nachricht einen errechneten MAC (*message authentication code* – Datensicherungscode) nach und sendet beide Teile zum Empfänger. Dieser kann über die Nachricht nun selbst einen MAC berechnen und vergleicht ihn mit dem erhaltenen MAC. Stimmen beide überein, wurde die Nachricht während der Übertragung nicht verändert.

Bild 4.35 Die übliche Anordnung von Nachricht und Message Authentication Code (MAC).

Um einen MAC zu bilden, verwendet man einen Kryptoalgorithmus mit einem geheimen Schlüssel, der beiden Kommunikationspartnern bekannt sein muss. Im Prinzip ist ein MAC eine Art von Fehlererkennungscode (*error detection code – EDC*), der allerdings nur überprüft werden kann, wenn man den geheimen Schlüssel dazu kennt. Deshalb existiert parallel zur Bezeichnung MAC, neben einigen anderen Begriffen, auch noch die Bezeichnung kryptografische Prüfsumme (*cryptographic checksum – CCS*), welche aber technisch gesehen völlig identisch mit einem MAC ist. Generell besteht der Unterschied zwischen den beiden Bezeichnungen darin, dass der Begriff MAC bei Datenübertragungen verwendet wird und CCS bei allen anderen Einsatzgebieten. Oft findet man auch noch den Begriff „Signatur" als Ersatzwort für MAC. Es gibt allerdings einen Unterschied zu dem Begriff „digitale Signatur", welcher darin besteht, dass diese mit einem asymmetrischen Kryptoalgorithmus zu berechnen ist.

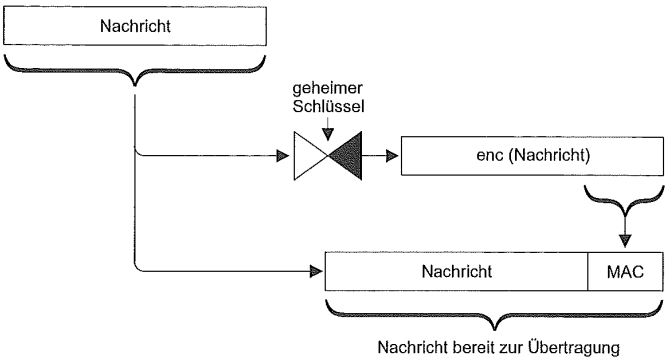


Bild 4.36 Beispiel für den Ablauf bei einer MAC-Berechnung.

Zur Berechnung eines MAC kann prinzipiell jeder Kryptoalgorithmus benutzt werden. In der Praxis benutzt man allerdings fast nur den DES, der auch hier zur Demonstration des Verfahrens dient.

Durch die DES-Verschlüsselung einer Nachricht im CBC-Modus sind alle nachfolgenden Blöcke mit den vorangehenden verknüpft. Damit ist der letzte Block abhängig von allen vorangehenden. Dieser Block oder ein Teil davon stellt den MAC einer Nachricht dar. Die Nachricht selber bleibt aber im Klartext, wird also nicht verschlüsselt übertragen.

Es existieren bei der MAC-Bildung mit dem DES-Algorithmus noch einige wichtige Rahmenbedingungen. Falls die Länge der Nachricht kein Vielfaches von acht Byte ist, muss sie dahingehend erweitert werden. Dies fällt allgemein unter den Begriff Padding. Allerdings findet hier meistens nur ein Auffüllen mit '00' statt (ANSI X 9.9 – Message Authentication). Dies ist hier erlaubt, da die Länge und Position des MAC innerhalb der Nachricht vorher vereinbart sein muss. Der MAC selber bildet sich aus den linken (MSB) vier Byte des letzten im CBC-Modus verschlüsselten Blockes. Die Paddingbytes werden allerdings bei einer Nachrichtenübertragung nicht mit übertragen. Damit minimieren sich die zu übertragenden Daten auf die zu schützenden Nutzdaten und den nachgestellten MAC.

4.8 Schlüsselmanagement

Alle Managementprinzipien, wie mit Schlüsseln für kryptografische Algorithmen verfahren werden soll, haben nur den einen Zweck, dass im Falle des Bekanntwerdens eines oder mehrerer dieser geheimen Schlüssel die Auswirkungen auf das System und die Chipkarten-Anwendung so gering wie möglich gehalten werden. Könnte man sicher sein, dass die Schlüssel immer geheim bleiben, dann würde pro Chipkarte ein einziger geheimer Schlüssel genügen. Doch diese Sicherheit der Geheimhaltung kann niemand garantieren.

Durch die nachfolgend beschriebenen Prinzipien zur Erhöhung der Sicherheit bei Verwendung von Schlüsseln für kryptografische Algorithmen wächst die Anzahl der Schlüssel stark an. Sind in einer Chipkarte alle hier beschriebenen Verfahren und Prinzipien realisiert, beanspruchen die Schlüssel meist mehr als die Hälfte des Speicherplatzes der Nutzdaten.

Doch müssen je nach Einsatzgebiet nicht alle Verfahren und Prinzipien Anwendung finden. Es besteht beispielsweise keine starke Notwendigkeit, mehrere Schlüsselgenerationen zu unterstützen, wenn die Chipkarte nur eine kurze Gültigkeitsdauer hat, da dies den Mehraufwand an Verwaltung und Speicher nicht rechtfertigen würde.

4.8.1 Abgeleitete Schlüssel

Da Chipkarten im Gegensatz zu den Terminals von jedermann mit nach Hause genommen und dort eventuell mit großem Aufwand analysiert werden können, sind sie natürlich den stärksten Angriffen ausgesetzt. Wenn keine Hauptschlüssel (*master key*) in der Karte vorhanden sind, lassen sich die Auswirkungen im Fall eines erfolgreichen Auslesens minimieren. Deshalb befinden sich in der Chipkarte nur die von einem Hauptschlüssel abgeleiteten Schlüssel.

Der abgeleitete Schlüssel (*derived key*) wird mit einem kryptografischen Algorithmus erzeugt. Die Eingangswerte dafür sind ein kartenindividuelles Merkmal und ein Hauptschlüssel. Als Kryptoalgorithmus dient üblicherweise der Triple-DES oder AES. Das individuelle Merkmal ist der Einfachheit halber die Kartennummer. Diese ist eine bei der Herstellung in die Karte geschriebene Nummer, die im gesamten System einzigartig ist und zur systemweiten Identifikation der Chipkarte benutzt werden kann.

Ein abgeleiteter Schlüssel ist somit ein Unikat und kann beispielsweise mit der folgenden Funktion erzeugt werden:

abgeleiteter Schlüssel = enc *(Hauptschlüssel; Kartennummer)*

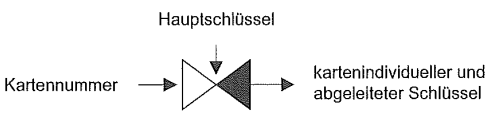


Bild 4.37 Eine Variante, wie ein abgeleiteter kartenindividueller symmetrischer Schlüssel mithilfe der Kartennummer und einem Hauptschlüssel erzeugt werden kann.

4.8.2 Schlüsseldiversifizierung

Um im Kompromittierungsfall eines Schlüssels die Auswirkungen so weit wie möglich zu minimieren, verwendet man oft für jede kryptografische Funktion eigene Schlüssel. So kann man beispielsweise zwischen Schlüsseln für Signaturen, gesicherte Datenübertragung, Authentisierung und Verschlüsselung unterscheiden. Für jeden Schlüssel muss es demnach einen eigenen Hauptschlüssel geben, mit dem die individuellen Schlüssel abgeleitet werden.

4.8.3 Schlüsselversionen

Es reicht im Regelfall nicht aus, die gesamte Lebensdauer einer Chipkarte mit einer einzigen Schlüsselgeneration zu bestreiten. Man denke sich nur den Fall, dass durch einen Angriff ein Hauptschlüssel berechnet werden konnte. Dann müssten alle Anwendungsanbieter ihre Systeme stilllegen und der Kartenherausgeber alle Chipkarten austauschen. Der dabei

verursachte Schaden wäre immens. Deshalb besitzen alle modernen Systeme die Möglichkeit, auf neue Schlüsselgenerationen weiterzuschalten.

Das Umschalten kann im schlimmsten Fall durch die Kompromittierung eines Schlüssels erzwungen werden oder routinemäßig in einem festen oder variablen Zeitraster geschehen. Das Ergebnis ist, dass alle Schlüssel im System durch neue ausgetauscht worden sind, ohne dass eine Rückrufaktion für die Chipkarten notwendig geworden wäre. Da sich die Hauptschlüssel in den Terminals und den Systemkomponenten darüber befinden, bedarf es nur eines gesicherten Datenaustausches, um diese mit den neuen, noch unbekannten Hauptschlüsseln zu versorgen.

4.8.4 Dynamische Schlüssel

Für viele Anwendungen, gerade im Bereich der Sicherung der Datenübertragung, ist es Stand der Technik, dynamische Schlüssel zu verwenden. Andere Bezeichnungen dafür sind temporäre Schlüssel, Sitzungsschlüssel oder Session Keys. Dazu erzeugt zuerst einer der beiden Kommunikationsteilnehmer eine Zufallszahl oder einen anderen für diese Sitzung verwendeten Wert und teilt sie dem zweiten Teilnehmer mit. Nun teilt sich das Verfahren auf, je nachdem ob nur symmetrische oder auch asymmetrische Kryptoalgorithmen zum Einsatz kommen.

Dynamische Schlüssel bei symmetrischen Kryptoalgorithmen
Die von einem der beiden Teilnehmer erzeugte Zufallszahl wird bei Verfahren, die nur symmetrische Kryptoalgorithmen benutzen, als Klartext zum Gegenüber gesendet. Chipkarten und Terminal verschlüsseln diese dann mit einem abgeleiteten Schlüssel. Das Ergebnis ist ein nur für diese eine Sitzung geltender Schlüssel.

dynamischer Schlüssel = enc *(abgeleiteter Schlüssel; Zufallszahl)*

Der große Vorteil von dynamischen Schlüsseln liegt darin, dass sie sich von Sitzung zu Sitzung ändern und damit einen Angriff wesentlich erschweren. Vorsicht ist aber mit diesen Schlüsseln geboten, wenn damit Signaturen erzeugt werden, da zur Überprüfung dieser Signatur natürlich der temporäre Schlüssel benötigt wird. Dieser Schlüssel lässt sich aber nur mit der gleichen Zufallszahl wie beim Signieren erzeugen. Daraus folgt, dass bei Verwendung eines temporären Schlüssels immer auch die den Schlüssel erzeugende Zufallszahl für die Überprüfung bereitgehalten und damit gespeichert werden muss.

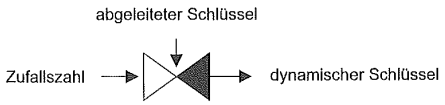


Bild 4.38 Eine Möglichkeit der Erzeugung eines dynamischen symmetrischen Schlüssels mithilfe einer Zufallszahl und einem abgeleiteten symmetrischen Schlüssel.

Die ANSI Norm X 9.17 sieht für die Erzeugung von dynamischen und abgeleiteten Schlüsseln ein etwas aufwendigeres Verfahren als vorhergehend beschrieben vor. Es findet aber im Bereich des Zahlungsverkehrs breite Anwendung. Man benötigt dazu einen zeit- oder sitzungsabhängigen Wert T_i sowie einen Schlüssel Key_{Gen}, der für die Generierung

der neuen Schlüssel reserviert ist. Anschließend kann man mit dem Startschlüssel Key_i beliebige neue Schlüssel berechnen. Dieses Schlüsselerzeugungsverfahren hat den zusätzlichen Vorteil, dass es nicht rückrechenbar ist, also eine Einwegfunktion darstellt.

$$Key_{i+1} := enc(Key_{Gen}; enc(Key_{Gen}; (T_i \text{ XOR } Key_i)))$$

Dynamischer Schlüsselaustausch mithilfe von asymmetrischen Kryptoalgorithmen
Die folgenden beiden Bilder zeigen die Erzeugung und den darauf folgenden Austausch eines symmetrischen dynamischen Schlüssels zur Verschlüsselung einer Nachricht. Zum Schlüsselaustausch wird ein asymmetrischer Kryptoalgorithmus, wie RSA oder DSS, benutzt.

Ein ähnliches Verfahren findet beispielsweise bei PGP Anwendung. Dort werden RSA und IDEA als Kryptoalgorithmen benutzt. Dieses hybride Verfahren hat grundsätzlich den Vorteil, dass die eigentliche Massendatenverschlüsselung mit einem symmetrischen Kryptoalgorithmus durchgeführt werden kann, welcher einen wesentlich höheren Durchsatz hat als asymmetrische Kryptoalgorithmen.

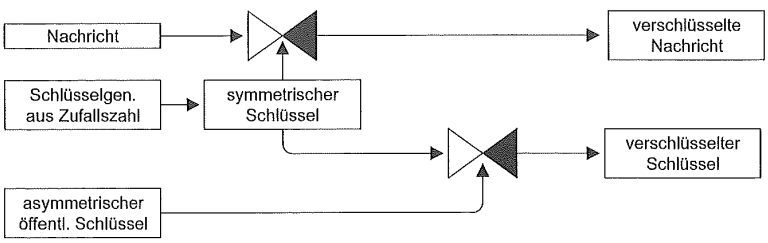


Bild 4.39 Beispiel für den Ablauf des Schlüsselaustausches mit der Kombination von asymmetrischen und symmetrischen kryptografischen Algorithmen. Das Bild zeigt die Erzeugung eines verschlüsselten dynamischen symmetrischen Schlüssels, der dann unter Zuhilfenahme eines asymmetrischen Kryptoalgorithmus zwischen zwei Instanzen ausgetauscht wird. Die Erzeugung und der Austausch des Schlüsselpaars für den asymmetrischen Kryptoalgorithmus wurde separat und vorab durchgeführt und sind hier nicht aufgezeigt.

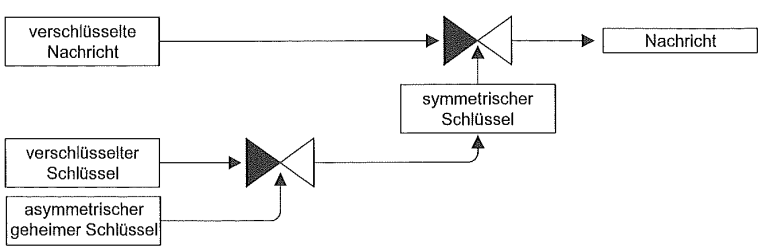


Bild 4.40 Beispiel für den Ablauf des Schlüsselaustausches mit der Kombination von asymmetrischen und symmetrischen kryptografischen Algorithmen. Das Bild zeigt die Rückgewinnung eines zuvor verschlüsselten dynamischen symmetrischen Schlüssels unter Zuhilfenahme eines asymmetrischen Kryptoalgorithmus. Die Erzeugung und der Austausch des Schlüsselpaars für den asymmetrischen Kryptoalgorithmus wurden separat und vorab durchgeführt und sind hier nicht aufgezeigt worden.

4.8.5 Schlüsselinformationen

Um nun die Schlüssel innerhalb der Chipkarte von außen adressieren zu können, bedarf es eines möglichst einfachen Mechanismus. Das Betriebssystem der Chipkarte muss zusätzlich in jedem Fall sicherstellen, dass die Schlüssel nur für den ihnen zugedachten Verwendungszweck benutzt werden können. Es verhindert also, dass z. B. ein Authentisierungsschlüssel für die Verschlüsselung von Daten benutzt werden kann. Zusätzlich zu dem Verwendungszweck benötigt man zur Adressierung des Schlüssels noch die Schlüsselnummer. Diese Nummer ist die eigentliche Referenz auf den Schlüssel. Des Weiteren ist auch die Versionsnummer notwendig, um so den konkreten Schlüssel adressieren zu können.

Tabelle 4.21 Typische Informationen über einen Schlüssel, die in einer Chipkarte gespeichert werden.

Datenelement	Kommentar
Schlüsselnummer	Dies ist innerhalb der Schlüsseldatei eine eindeutige Nummer zur Referenzierung des Schlüssels.
Versionsnummer	Dies ist eine Versionsnummer des Schlüssels, welche beispielsweise Einfluss auf die Schlüsselableitung haben kann.
Verwendungszweck	Dieses Datenelement gibt an, für welche kryptografischen Algorithmen und Verfahren der Schlüssel benutzt werden darf.
Sperrvermerk	Damit kann der Schlüssel temporär oder irreversibel zur Benutzung gesperrt werden.
Fehlbedienungszähler	Dies ist ein Zähler, welcher nicht erfolgreiche Benutzungen des Schlüssels im Rahmen kryptografischer Verfahren protokolliert.
Maximalwert des Fehlbedienungszählers	Ist der Maximalwert des Fehlbedienungszählers erreicht, so ist die Verwendung des Schlüssels gesperrt.
Länge des Schlüssels	Die Länge des Schlüssels.
Schlüssel	Der Schlüssel selber.

Einige Betriebssysteme für Chipkarten sehen vor, dass bei einer fehlgeschlagenen Aktion mit einem Schlüssel, z. B. bei der Authentisierung, ein Fehlbedienungszähler am Schlüssel erhöht wird. Damit kann man das Ausspähen eines Schlüssels durch Probieren recht zuverlässig verhindern, obwohl dies aufgrund der langen Rechenzeiten in der Chipkarte kein ernst zu nehmender Angriff ist. Erreicht der Fehlbedienungszähler den Maximalwert, so ist der Schlüssel gesperrt und kann nicht mehr verwendet werden. Der Fehlbedienungszähler wird auf null zurückgesetzt, wenn die Aktion erfolgreich verlaufen ist. Allerdings muss solch ein Mechanismus sehr behutsam eingesetzt werden, da es sehr leicht aufgrund eines falschen Hauptschlüssels in einem Terminal zu Massenausfällen an Chipkarten kommen kann. Das Rücksetzen des Fehlbedienungszählers wird im Regelfall nur an einem speziellen Terminal unter Überprüfung der Identität des Kartenbesitzers vorgenommen.

Manche Systeme verbieten die Weiterverwendung der alten Schlüsselversionen. Dazu stattet man den Schlüssel mit einem Sperrvermerk aus, der aktiviert wird, sobald ein neuer Schlüssel mit gleicher Schlüsselnummer angesprochen wird.

4.8.6 Beispiel für Schlüsselmanagement

Im Folgenden wird beispielhaft ein mögliches Schlüsselmanagement für ein auf Chipkarten aufbauendes System dargestellt, um überblickshaft die eben vorgestellten Prinzipien an einem Gesamtbeispiel nochmals zu verdeutlichen. Große reale Systeme sind im Vergleich dazu oft noch viel komplexer und mehrstufiger aufgebaut. Bei kleineren Systemen gibt es häufig überhaupt keine Schlüsselhierarchien, da ein globaler geheimer Schlüssel für alle Karten benutzt wird. Das vorgestellte System stellt ein Mittelmaß zwischen sehr einfach aufgebauten Systemen und großen Systemen dar und ist aus diesem Grund auch sehr anschaulich.

Im vorgestellten Beispiel könnten die Schlüssel zum Laden und Bezahlen mit einer elektronische Geldbörse verwendet werden und symmetrische Kryptografie benutzen. Es sind in jedem Fall Schlüssel, die im System wichtig sind, da sie durch die beschriebene Schlüsselhierarchie relativ gut geschützt sind. Die einzelnen Ableitungsfunktionen sind hier nicht genauer erläutert, man könnte aber in allen Fällen den DES oder den Triple-DES verwenden. Es wird auch nicht genauer auf die Länge der Schlüssel eingegangen, doch kann sie durchaus variieren. Die am weitesten in der Hierarchie oben stehenden Schlüssel werden normalerweise aus sicherheitstechnischen Gründen mit kryptografisch stärkeren Funktionen abgeleitet als die weiter unten stehenden.

Der Schlüssel, der in der Hierarchie am höchsten steht, ist der generelle Hauptschlüssel. Er existiert nur einmal für eine Generation von Schlüsseln. Eine Generation könnte beispielsweise für ein Jahr gelten, im darauf folgenden Jahr würde auf eine neue Generation, d. h. auf einen neuen generellen Hauptschlüssel umgeschaltet. Dieser Schlüssel hat die höchste Sensitivität in Bezug auf Sicherheit. Würde er bekannt, sind alle Schlüssel für diese Generation berechenbar, und das System ist für eine Schlüsselgeneration gebrochen. Der Hauptschlüssel kann durch eine Zufallszahl erzeugt werden. Vorstellbar ist auch, den Hauptschlüssel aus den Würfelergebnissen von mehreren unabhängigen Personen aufzubauen, die damit jeweils nur einen Teil des Schlüssels kennen. Er sollte nie einer einzelnen Person bekannt sein, und die Erzeugung darf unter keinen Fällen nachvollziehbar sein.

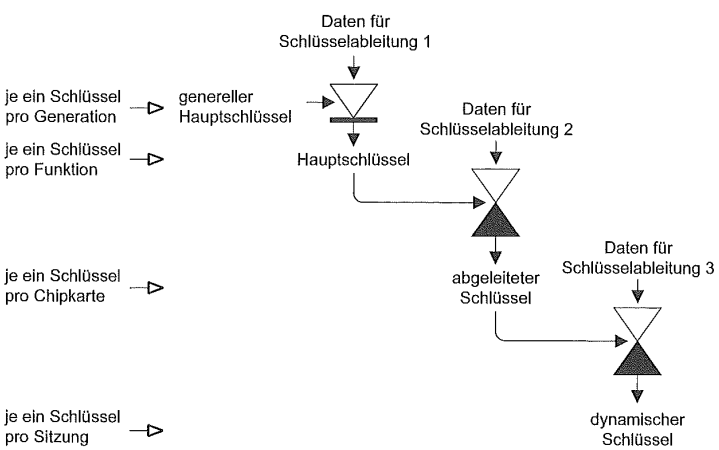


Bild 4.41 Beispiel für eine Schlüsselhierarchie in einem auf Chipkarten aufbauenden System mit symmetrischen Kryptoalgorithmen.

Mit diesem generellen Hauptschlüssel werden nun auf Funktionen aufgetrennt die eigentlichen Hauptschlüssel abgeleitet. Die Funktion eines Schlüssels kann das Laden oder das Bezahlen mit einer elektronischen Geldbörse sein. Dabei wird hier im Beispiel zur Ableitung der nach Funktionen aufgetrennten Hauptschlüssel eine Einwegfunktion (z. B. modifizierter Triple-DES) benutzt. Diese wird verwendet, um eine Rückrechnung von einem Hauptschlüssel zum generellen Hauptschlüssel zu unterbinden. Würde trotz aller Vorsichtsmaßnahmen ein Hauptschlüssel bekannt und keine Einwegfunktion zur Ableitung verwendet, wäre es in Kenntnis der Ableitungsdaten möglich, den generellen Hauptschlüssel zu berechnen. Der Grund für die Einwegfunktion an dieser Stelle ist die Annahme, dass in dem fiktiven Börsensystem die Hauptschlüssel in den Sicherheitsmodulen der Terminals vor Ort sind. Damit sind sie vom Standpunkt der Systemsicherheit her wesentlich mehr gefährdet als die generellen Hauptschlüssel, die das Hintergrundsystem nie verlassen.

Die nächste Stufe in der Schlüsselhierarchie sind die abgeleiteten Schlüssel. Dabei handelt es sich um die in den Chipkarten befindlichen Schlüssel. In jeder Karte befindet sich ein nach Generation und Funktion getrennter Satz von abgeleiteten Schlüsseln. Wird nun eine solche Karte an einem Terminal benutzt, kann dieses anhand der Daten für die Schlüsselableitung den jeweiligen abgeleiteten Schlüssel berechnen. Die Daten zur Ableitung werden selbstverständlich vorab auf der Karte vom Terminal ausgelesen. Mit dem abgeleiteten Schlüssel kann nun in einem weiteren Schritt der dynamische Schlüssel berechnet werden, der sitzungsindividuell ist. Er ist also nur über die Zeitdauer einer Sitzung gültig. Dies sind bei typischen Chipkarten-Anwendungen minimal einige hundert Millisekunden und maximal einige Sekunden. Nach der Sitzung wird der Schlüssel nicht mehr verwendet.

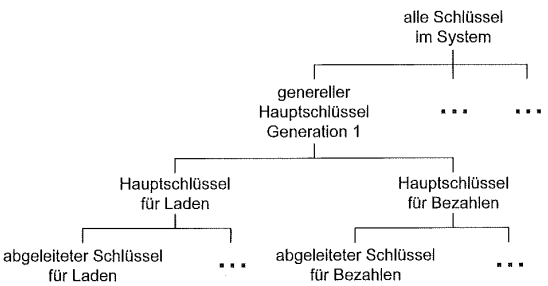


Bild 4.42 Beispiel für Schlüssel in einem System einer elektronischen Geldbörse mit den Funktionen Laden und Bezahlen. Es sind nur gespeicherte Schlüssel aufgeführt, d. h., die dynamisch erzeugten Schlüssel für die einzelnen Sitzungen wurden aus Vereinfachungsgründen weggelassen.

Das vorgestellte System mag auf den ersten Blick aufwendig und kompliziert erscheinen, doch ist dies im Vergleich zu realen Systemen nicht der Fall. Der Zweck eines solchen Systems ist, einen genauen Weg vorzugeben, wie alle Schlüssel in einem System zu erzeugen sind. Auch zeigt es implizit auf, welche Maßnahmen ergriffen werden müssen, falls einmal ein Schlüssel bekannt werden sollte. Ist es der generelle Hauptschlüssel, muss auf eine neue Generation umgeschaltet werden, damit das System wieder ohne Sicherheitsbedenken betrieben werden kann. Wird dagegen ein Hauptschlüssel bekannt, ist nur

der darunter liegende Zweig zu sperren oder auf eine neue Generation umzuschalten. Ist es ein abgeleiteter Schlüssel, der nicht mehr geheim ist, so erfordert es als einzige Maßnahme die Sperrung der jeweiligen Karte. Weitere Eingriffe in das Schlüsselmanagement wären dann wohl übertrieben. Alle diese Maßnahmen setzen natürlich voraus, dass der Grund für das Bekanntwerden eines oder mehrerer Schlüssel ermittelt werden konnte und Letzteres in Zukunft verhindert wird.

Aufgrund dieser Schlüsselhierarchie müssen natürlich sehr viele Schlüssel erzeugt und in den Chipkarten gespeichert werden. Es besteht natürlich immer die Möglichkeit, verschiedene Funktionen auf einen Schlüssel zusammenzulegen, um auf diese Weise Speicherplatz zu sparen. Auch ist durchaus ein anderer Aufbau der Schlüsselhierarchie denkbar. Dies hängt naturgemäß sehr stark vom System ab, für das das Schlüsselmanagement entwickelt wurde.

4.9 Hash-Funktionen

Selbst leistungsfähige Computer benötigen zur Berechnung einer digitalen Signatur sehr viel Zeit. Bei größeren Dokumenten wären mehrere Signaturen erforderlich, da die Länge eines zu signierenden Dokuments nicht beliebig sein kann. Deshalb behilft man sich mit einem Trick. Man komprimiert zuerst das Dokument auf eine viel geringere Länge, die fest ist, und berechnet dann die digitale Signatur über die komprimierten Daten. Dabei ist es nicht entscheidend, ob die Komprimierung wieder rückgängig gemacht werden kann, da sie mit dem Originaldokument jederzeit wieder nachvollziehbar ist. Funktionen für diese Berechnung nennt man Einweg-Hash-Funktionen.

Allgemein ausgedrückt, ist also eine Einweg-Hash-Funktion eine Funktion, die aus einem Dokument mit variabler Länge einen Wert mit fester Länge berechnet, der den Inhalt des Dokuments in komprimierter und nicht rückrechenbarer Form wiedergibt. Benutzt werden diese Funktionen im Chipkartenbereich ausschließlich zur Berechnung von Eingangswerten für digitale Signaturen. Entspricht das Dokument in seiner Länge nicht dem Vielfachen der von der Hash-Funktion verwendeten Blocklänge, muss es entsprechend gepadded werden.

Damit eine Hash-Funktion gut ist, muss sie mehrere Eigenschaften aufweisen. Das Ergebnis sollte eine feste Länge haben, damit es optimal von den Signaturalgorithmen weiterverwendet werden kann. Da normalerweise große Datenmengen verarbeitet werden müssen, muss die Hash-Funktion auch über einen großen Durchsatz verfügen. Es muss also einfach sein, den Hashwert zu berechnen. Andererseits sollte es schwierig, besser unmöglich sein, aus einem vorhandenen Hashwert das ursprüngliche Dokument zu berechnen. Schließlich muss die Hash-Funktion kollisionsresistent sein. Dies bedeutet, dass es nicht einfach sein darf, für ein vorgegebenes Dokument ein zweites verändertes Dokument zu finden, das den gleichen Hashwert hat.

Allerdings steht natürlich zweifelsfrei fest, dass es noch weitere Dokumente geben muss, für die der gleiche Hashwert existiert. Schließlich werden alle möglichen Nachrichten mit jeder Länge zwischen null und unendlich auf einen Hashwert mit fester Länge abgebildet. Dies hat zwangsläufig zur Folge, dass Kollisionen auftreten. Deshalb wird auch der Begriff kollisionsresistent und nicht kollisionsfrei verwendet.

Wie würde sich nun eine Kollision auswirken? Es gäbe dann also zwei verschiedene Dokumente mit dem gleichen Hashwert und insofern mit der gleichen digitalen Signatur. Die Auswirkungen wären fatal: Die Signatur wäre wertlos, da man am signierten Dokument Änderungen vornehmen könnte, ohne dass diese erkannt würden. Genau dies ist der Fall bei einem der beiden typischen Angriffe, die bei Hash-Funktionen vorstellbar sind. Man sucht systematisch ein zweites Dokument, das denselben Hashwert besitzt wie das ursprüngliche. Wenn das Dokument auch inhaltlich einen Sinn ergibt, hat man die auf den Hashwert erfolgte digitale Signatur diskreditiert. Da aber in diesem Fall beide Dokumente austauschbar sind, ist die Signatur wertlos. Es ist schließlich ein gewaltiger Unterschied, ob ein Kaufvertrag über ein Haus sich auf eine Höhe von 10000 € oder 750000 € bezieht.

Der zweite Angriff auf einen Hashwert ist etwas hintergründiger. Man erstellt von vornherein zwei Dokumente, die den gleichen Hashwert besitzen, aber inhaltlich unterschiedlich sind. Dies ist nicht weiter schwierig, wenn man an die zur Verfügung stehenden Sonderzeichen und Erweiterungen des Zeichensatzes denkt. Die Folge: Eine digitale Signatur ist für beide Dokumente gültig; man kann nicht mehr beweisen, welches Dokument ursprünglich unterschrieben wurde.

Das Auffinden zweier Dokumente, die zum gleichen Hashwert führen, ist nicht so schwierig, wie man zunächst meint. Man macht sich dabei das aus der Statistik bekannte Geburtstags-Paradoxon zunutze. Dieses dreht sich um die Frage, wie viele Menschen in einem Raum sein müssen, damit die Wahrscheinlichkeit größer als 50 % ist, dass einer der Anwesenden am gleichen Tag Geburtstag hat wie der Fragesteller. Die Antwort ist einfach: Man muss nur den Geburtstag des Fragestellers mit den Geburtstagen aller im Raum befindlichen Personen vergleichen. Es müssen mindestens 183 (= 365/2) Personen sein.

Die nächste Frage offenbart das Paradoxe oder besser das Überraschende an diesen Geburtstagsvergleichen. Wie viele Menschen müssen in einem Raum sein, damit die Wahrscheinlichkeit, dass zwei Personen am gleichen Tag Geburtstag haben, größer als 50 % ist. Die überraschende Lösung besteht darin, dass es nur 23 Personen sein müssen. Der Grund dafür liegt in der Tatsache, dass zwar nur 23 Personen im Raum sind, insgesamt aber 253 Paare sich zum Geburtstagsvergleich bilden lassen. Die Wahrscheinlichkeit, dass zwei Personen am gleichen Tag geboren sind, wird dann auf der Grundlage dieser Paarbildung berechnet.

Genau dieses Paradoxon macht man sich beim Angriff auf eine Hash-Funktion zunutze. Es ist viel leichter, zwei Dokumente zu erstellen, die einen gemeinsamen Hashwert bilden, als ein Dokument so lange abzuändern, bis es einem vorgegebenen Hashwert entspricht. Die Auswirkung dieses Angriffes ist, dass die Ergebnisse von Hash-Funktionen genügend groß sein müssen, um die beiden beschriebenen Angriffe erfolgreich zu verhindern. So erzeugen die meisten Hash-Funktionen einen Ausgangswert von mindestens 128 Bit Länge, welcher zurzeit auch allgemein als ausreichend gegen obige Form von Angriffen gesehen wird.

Mittlerweile wurden viele Hash-Funktionen publiziert, einige davon sind auch in Normen festgelegt. Allerdings kommt es immer wieder zu Änderungen in bestehenden Hash-Funktionen, wenn erfolgreiche Angriffe bekannt wurden. Im Folgenden ein kurzer Überblick, der die derzeit üblichen Hash-Funktionen aufzeigt. Auf die interne Funktionsweise kann hier aus Platzgründen leider nicht eingegangen werden.

In der ISO/IEC 10118-2 ist eine Hash-Funktion genormt, die auf einem n-Bit-Blockverschlüsselungsalgorithmus (z. B. DES) aufbaut. Die Größe des Hashwertes kann

mit dem beschriebenen Algorithmus n Bit oder 2 · n Bit lang sein. Die mittlerweile sehr selten eingesetzte Hash-Funktion MD4 (*message digest 4*) und die Weiterentwicklung MD5 wurde von Ronald L. Rivest in den Jahren 1990/1991 veröffentlicht. Sie basieren auf einem eigenständigen Algorithmus und produzieren beide einen 128 Bit langen Hashwert. Das NIST veröffentlichte Anfang 1992 den SHA als eine Hash-Funktion für den DSS. Nach Bekanntwerden einiger Schwächen wurde er überarbeitet, und das Ergebnis ist seit Mitte 1995 als SHA-1 bekannt und als FIPS 180-1 auch genormt.

Tabelle 4.22 Überblick und Zusammenfassung von Hash-Funktionen

Name	Größe der Eingangsblöcke	Größe des Hashwerts
ISO/IEC 10 118-2	n Bit (z. B. 64, 128 Bit)	n/2 · n Bit (z. B. 64, 128 Bit)
MD4	512 Bit	128 Bit
MD5	512 Bit	128 Bit
MDC-4	64 Bit	128 Bit
RIPEMD-128	512 Bit	128 Bit
RIPEMD-160	512 Bit	160 Bit
SHA-1	512 Bit	160 Bit

Da die Datenübertragung zur Chipkarte in den meisten Fällen langsam ist, wird die Hash-Funktion im Terminal oder in einem daran angeschlossenen Computer ausgeführt. Dieser Nachteil wird durch die auf diese Weise möglich gewordene Austauschbarkeit der Hash-Funktion wettgemacht. Außerdem wäre es in den meisten Fällen aus Speicherplatzgründen nicht möglich, eine Hash-Funktion in einer Chipkarte unterzubringen, da sich der Programmaufwand fast in jedem Fall im Bereich von 4 kByte Assemblercode bewegt. Der Durchsatz von typischen Hash-Funktionen ist entsprechend der Anforderung sehr hoch und liegt meist mindestens auf einem 80386 Computer mit 33 MHz im Bereich von 300 kByte/s, auf einem Pentium PC mit 200 MHz in der Größenordnung von 4 bis 8 MByte/s.

4.10 Zufallszahlen

In Verbindung mit kryptografischen Verfahren ist es immer wieder notwendig, Zufallszahlen zu benutzen. Typische Einsatzgebiete im Chipkartenbereich sind dabei die Sicherstellung der Einzigartigkeit einer Sitzung bei Authentisierung, als Füller (*padding*) bei Verschlüsselungen oder als Startwert bei Sendefolgezählern. Die Länge der dafür benötigten Zufallszahlen bewegt sich meist zwischen 2 und 8 Bytes. Die maximale Länge von acht kommt natürlich von der Blockgröße des DES-Algorithmus.

Die Sicherheit aller dieser Verfahren setzt Zufallszahlen voraus, die nicht vorhersehbar oder von außen beeinflussbar sind. Ideal wäre ein auf Hardware aufbauender Zufallszahlengenerator im Mikrocontroller der Chipkarte. Dieser müsste aber vollständig unabhängig von äußeren Einflüssen, wie Temperatur, Versorgungsspannung, Strahlung o. Ä. sein, da man ihn sonst manipulieren könnte. Auf diese Weise wäre es möglich, einige Verfahren,

deren Sicherheit auf der Zufälligkeit der Zufallszahlen aufbaut, zu kompromittieren. Die zurzeit existierenden Zufallszahlengeneratoren auf Chipkarten-Mikrocontrollern basieren in der Regel auf spannungsgesteuerten Oszillatoren und daran angehängten linear rückgekoppelten Schieberegistern (*linear feedback shift register – LFSR*).

Auch beim derzeitigen Stand der Technik ist es schwierig, einen guten und von äußeren Einflüssen unabhängigen Zufallszahlengenerator auf dem Silizium des Mikrocontrollers aufzubauen (TRNG – *true random number generator*). Deshalb behelfen sich die Betriebssystemdesigner oftmals mit Realisierungen in Software. Das Ergebnis sind Pseudozufallszahlengeneratoren (PRNG – *pseudo random number generator*), die meist auch sehr gute (d. h. zufällige) Zufallszahlen generieren. Diese sind aber keine echten Zufallszahlen, da sie mithilfe eines streng deterministischen Algorithmus berechnet werden und damit bei Kenntnis des Algorithmus und seiner Eingangswerte auch vorhersagbar sind. Daher bezeichnet man sie als Pseudozufallszahlen.

Von großer Bedeutung ist auch, dass alle Chipkarten einer Produktionsserie unterschiedliche Folgen von Zufallszahlen generieren, sodass nicht von der Zufallszahl einer Chipkarte auf die Zufallszahl einer anderen Chipkarte der gleichen Serie geschlossen werden kann. Dies wird dadurch realisiert, dass bei der Komplettierung des Chipkarten-Betriebssystems eine Zufallszahl als Startwert (*seed number*) des Zufallszahlengenerators in die Karte eingetragen wird.

4.10.1 Erzeugung von Zufallszahlen

Es gibt vielerlei Varianten, Zufallszahlen per Software zu erzeugen. Da jedoch in Chipkarten der verfügbare Speicherplatz stark beschränkt ist und die Rechenzeit zur Erzeugung einer Zufallszahl möglichst kurz sein soll, engt dies die Anzahl der Möglichkeiten ein. In der Praxis finden im Wesentlichen nur Verfahren Anwendung, die die ohnehin vorhandenen Funktionen des Betriebssystems nutzen und damit auch nur wenig zusätzlichen Programmcode beanspruchen.

Selbstverständlich darf die Unterbrechung einer Sitzung durch Reset oder Herausziehen der Karte aus dem Terminal die Güte der Zufallszahlen nicht beeinträchtigen. Auch muss der Generator so aufgebaut sein, dass nicht in jeder Sitzung die gleiche Folge von Zufallszahlen erzeugt wird. Dies klingt zwar trivial, erfordert aber zumindest einen Schreibzugriff auf das EEPROM, um so einen neuen Startwert für den Zufallszahlengenerator für die nächste Sitzung abzuspeichern. Das RAM ist dafür ja nicht geeignet, da es seinen Inhalt nur unter Spannung behalten kann. Ein Angriffsszenario bestünde nun darin, so lange Zufallszahlen zu erzeugen, bis der EEPROM-Speicher für den Startwert kaputtgeht. Dann müsste theoretisch in jeder Sitzung die gleiche Reihenfolge von Zufallszahlen auftreten; somit sind sie vorhersagbar und der Angreifer könnte einen Vorteil erringen. Wenn man den betreffenden EEPROM-Speicher als Ringpuffer aufbaut und bei einem Schreibfehler alle weiteren Aktionen sperrt, lässt sich dieser Angriff sehr leicht abwehren.

Ein weiteres sehr wichtiges Kriterium eines in Software implementierten Zufallszahlengenerators ist die Tatsache, dass dieser in keinem Fall in eine Endlosschleife läuft. Das Ergebnis wäre eine stark verkürzte Periodendauer der Wiederholung der Zufallszahlen. Dann wäre es ein Leichtes, sichere Vorhersagen zu treffen, und das System wäre gebrochen. In fast allen Chipkarten-Betriebssystemen ist zur Authentisierung ein Verschlüsselungsalgo-

rithmus vorhanden. Damit liegt die Idee nahe, auf diesem einen Zufallszahlengenerator aufzubauen. Dazu muss man wissen, dass ein guter Verschlüsselungsalgorithmus den Klartext möglichst gut durchmischt, sodass man vom Schlüsseltext nicht mehr ohne Kenntnis des Schlüssels auf den Klartext rückrechnen kann. Das auch als Avalanche-Kriterium bekannte Prinzip besagt, dass die Änderung eines Eingangsbits im Mittel die Hälfte aller Ausgangsbits ändern soll. Diese Eigenschaft lässt sich auch sehr gut für einen Zufallszahlengenerator nutzen. Wie der Generator genau aufgebaut ist, kann von Implementation zu Implementation verschieden sein.

Eine Möglichkeit dafür demonstriert Bild 4.43. Dieser Generator benutzt den DES-Algorithmus mit einer Blocklänge von 8 Byte, dessen Ausgangswert auf den Eingang rückgekoppelt ist. Es könnte natürlich auch jeder andere Verschlüsselungsalgorithmus dafür verwendet werden.

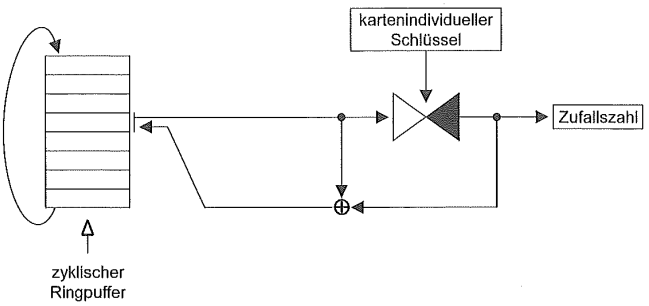


Bild 4.43 Beispiel für den Aufbau eines DES-Pseudozufallszahlengenerators für Chipkarten-Betriebssysteme. Der Generator ist primär auf geringe Schreibbelastung des EEPROMs ausgelegt.

Folgende Funktionsweise liegt dem Generator zugrunde: Der Wert eines Ringpufferelements wird mit dem DES und einem kartenindividuellen Schlüssel verschlüsselt. Der so erzeugte Schlüsseltext ist die 8 Byte lange Zufallszahl. Diese Zufallszahl, mit dem vorherigen Klartext mit XOR verknüpft, ergibt den neuen Eintrag im Ringpuffer des EEPROM. Anschließend wird auf den nächsten Eintrag im zyklisch aufgebauten Ringpuffer weitergeschaltet. Mathematisch gesehen, lassen sich die Abhängigkeiten wie folgt darstellen:

$$RND_n := \mathbf{f}(\text{Schlüssel}, RND_{(n-1)})$$

Bei der Komplettierung wird in jede Chipkarte der kartenindividuelle Schlüssel für den DES eingetragen und gleichzeitig der beispielsweise 12 · 8 Byte große Ringpuffer mit Zufallszahlen als Startwerte vorbelegt. Damit ist garantiert, dass jede Chipkarte eine eigene Reihenfolge von Zufallszahlen liefert. Der Ringpuffer verlängert die Lebensdauer des Generators um den Faktor 12. Nimmt man die garantierte Anzahl der Schreibzyklen eines EEPROM mit 100000 an, so können mit diesem Generator mindestens 1200000 Zufallszahlen mit einer Länge von 8 Byte erzeugt werden.

Das Löschen und Schreiben von 8 Byte in das EEPROM dauert ca. 2 · 2 · 3,5 ms und die Ausführung des DES bei 3,5 MHz etwa 17 ms, sofern der DES in Software realisiert wurde. Die restliche Bearbeitungszeit kann vernachlässigt werden. Somit benötigt die Chipkarte

für die gesamte Erzeugung einer Zufallszahl etwa 31 ms. Würde hingegen die DES-Berechnung in Hardware durchgeführt (typ. 0,1 ms/Block), könnte mit dem angegebenen Verfahren eine Zufallszahl in lediglich 14,4 ms erzeugt werden.

Ein anderes Beispiel eines Pseudozufallszahlengenerators zeigt Bild 4.44. Dieser wird nach jedem Reset der Chipkarte initialisiert, wobei auch der einzige EEPROM-Schreibzugriff stattfindet. Bei der darauf folgenden Erzeugung von Zufallszahlen wird nur noch auf das RAM zugegriffen, was den Generator relativ schnell macht. Von Nachteil dabei ist allerdings, dass man während der gesamten Sitzung einige Byte Speicher im RAM benötigt. Die statistische Qualität dieses Pseudozufallszahlengenerators ist nicht sehr gut, doch reicht sie für die übliche Verwendung bei Authentisierungsverfahren aus. Dort kommt es vor allem darauf an, dass keine Zufallszahlen mit kurzer Wiederholperiode erzeugt werden, da sich sonst die Authentisierung durch Wiedereinspielung von Nachrichten einer früheren Sitzung brechen ließe.

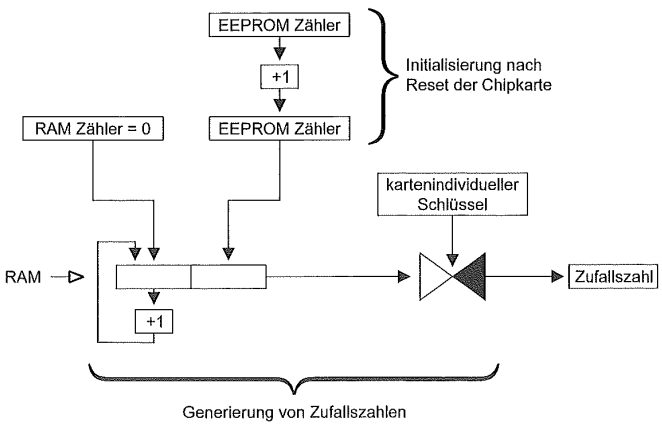


Bild 4.44 Beispiel für den Aufbau eines DES-Pseudozufallszahlengenerators für Chipkarten-Betriebssysteme. Dieser Generator ist schneller als der vorhergehende, da nur ein einziger EEPROM-Schreibvorgang pro Sitzung notwendig ist. Die Qualität der so erzeugten Zufallszahlen ist für die bei Chipkarten übliche Verwendung (Authentisierung durch das Challenge-Response-Verfahren) ausreichend.

Die FIPS 140-2 empfiehlt für Sicherheitsmodule, dass diese nach jedem Reset den eingebauten Zufallszahlengenerator mit statistischen Tests prüfen. Nur wenn diese erfolgreich durchlaufen worden sind, wird der Zufallszahlengenerator für die weitere Benutzung freigegeben. Die heute üblichen Chipkarten-Betriebssysteme besitzen selten eine solche Funktionalität, da man davon ausgeht, dass sich die Statistik der erzeugten Zufallszahlen aufgrund der deterministisch funktionierenden Pseudozufallszahlengeneratoren nicht wesentlich ändert.

Es gibt eine schier unüberschaubare Zahl von Vorschlägen, Normen und Implementationen von Pseudozufallszahlengeneratoren. Bekannte Beispiele dafür sind der Generator der X 9.17 Norm, der FIPS 186, die Vorschläge aus dem Internet Standard RFC 1750, die Aufstellungen bei Bruce Schneier [Schneier 96], Peter Gutmann [Gutmann 98 a] und Ben-

jamin Jun [Jun 99]. Der Leitgedanke bei einem Zufallszahlengenerator sollte immer sein, dass dieser so einfach und nachvollziehbar wie möglich ist. Nur dann lassen sich seine Eigenschaften beurteilen und infolgedessen auch die Qualität des Generators feststellen.

4.10.2 Prüfung von Zufallszahlen

Die von einem Zufallszahlengenerator erzeugten Zahlen müssen generell nach der Implementierung auf ihre Güte hin geprüft werden. Grundsätzlich sollte eine annähernde Gleichverteilung von Nullen und Einsen in den erzeugten Zufallszahlen herrschen. Es ist aber nicht damit getan, sich einige ausdrucken zu lassen und diese vergleichend zu prüfen. Der Test der Zufallszahlen kann mit den üblichen mathematischen Verfahren der Statistik durchgeführt werden. Dass dazu eine große Anzahl der 8 Byte langen Zufallszahlen notwendig ist, versteht sich von selbst. Um einigermaßen zuverlässige Aussagen machen zu können, sollten zwischen 10000 und 100000 Zufallszahlen erzeugt und ausgewertet werden. Die einzige Möglichkeit, diese Zahlen zu prüfen, besteht darin, computerunterstützte Prüfprogramme zu Hilfe zu nehmen.

Im Zusammenhang mit der Güte von Zufallszahlen ist eine Aussage über die Verteilung der erhaltenen Werte notwendig. Ist diese sehr ungleichmäßig und werden dadurch einige Werte stark bevorzugt, können genau diese Bereiche für eine Vorhersage genutzt werden. Es sollte also der Satz von Bernoulli so gut wie möglich erfüllt sein. Dieser besagt, dass das Auftreten einer Zahl unabhängig von ihrer Vorgeschichte nur von der Wahrscheinlichkeit des Auftretens der Zahl selber abhängen soll. So ist beispielsweise die Wahrscheinlichkeit einer 4 bei einem Sechsseitenwürfel immer 1/6, unabhängig davon, welche Zahl vorher gewürfelt wurde. Dies bezeichnet man auch als die Unabhängigkeit von Ereignissen.

Ebenfalls von großer Bedeutung ist die Periode der Zufallszahlen, d. h., nach wie vielen erzeugten Zufallszahlen sich die Reihenfolge wiederholt. Diese muss natürlich so groß wie möglich sein, auf jeden Fall aber größer als die Lebensdauer des Zufallszahlengenerators. Damit ist auf recht einfache und zuverlässige Weise ein Angriff durch Aufzeichnen aller Zufallszahlen einer Periode ausgeschlossen.

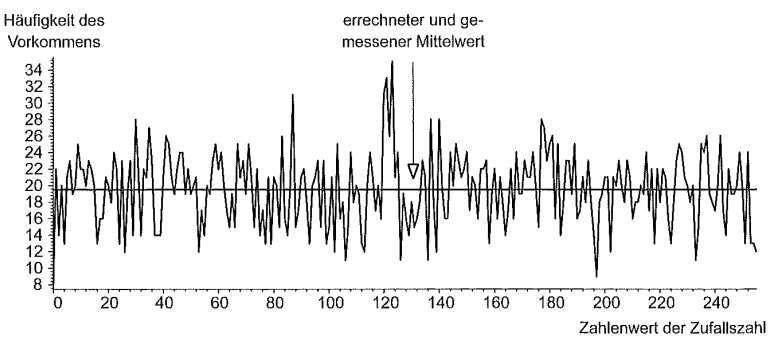


Bild 4.45 Statistische Verteilung einer Folge von 5 000 Zufallszahlen mit einer Länge von einem Byte. Dies bezeichnet man auch als Spektralverteilung über ein Byte. Die Zahlen wurden mit einem typischen Pseudozufallszahlengenerator einer Chipkarte erzeugt. Rechnerisch müsste jede der 256 möglichen (0 ... 255) Zufallszahlen 19,5-mal vorkommen.

In der Statistik gibt es eine beinahe unbegrenzte Anzahl von Tests, um die Zufälligkeit von Ereignissen zu untersuchen, doch kann man sich in der Praxis auf einige einfache und in der Aussage leicht zu interpretierende Tests beschränken. Es existieren im Übrigen eine Vielzahl von Veröffentlichungen zum Thema Test von Zufallszahlen [Knuth 97, Menezes 97] und auch entsprechende Normen [FIPS 140-2] und Standards [RFC 1750].

Ein sehr leicht zu erstellender und einfach zu interpretierender Test ist das Aufsummieren der Anzahl der vorkommenden Einzelbyte-Werte in einer größeren Anzahl von Zufallszahlen. Grafisch dargestellt, ergibt dies einen Überblick über die Verteilung der Zahlen. An der Abszisse trägt man die möglichen vorkommenden Werte auf und an der Ordinate die dazugehörige Häufigkeit des Vorkommens.

Sollen 8 Byte lange Zufallszahlen mithilfe dieses Diagramms untersucht werden, können an der Abszisse trotzdem nur ein Byte oder maximal zwei Bytes lange Werte aufgetragen werden, da sonst die Anzahl der für eine statistische Analyse notwendigen Zufallszahlen extrem groß wäre. Ein guter Richtwert ist, dass jede Zufallszahl ungefähr fünf- bis zehnmal pro Wert vorkommt, um eine einigermaßen gesicherte Aussage zu erhalten. Damit kann man einen schnellen Überblick gewinnen, ob die erzeugten Zufallszahlen die gesamte mögliche Bandbreite eines Bytes ausnutzen. Ist dies nicht der Fall und werden einige Werte sehr stark bevorzugt, dann ist dies für einen Angreifer zumindest ein erster Ansatzpunkt.

Leider sagt dieser Test nichts über die Reihenfolge der Vorkommnisse der Zufallszahlen aus, sondern nur etwas über die Verteilung. So wäre es denkbar, dass der Zufallszahlengenerator nur zyklisch von 0 bis 255 zählt. Damit wäre eine hervorragende Gleichverteilung erreicht, doch die Zahlen sind vollkommen vorhersagbar. Um auch dieses Kriterium der Güte von Zufallszahlen prüfen zu können, müssen andere Tests benutzt werden.

Ein ebenfalls aus der Praxis stammender Test, der für die einfache und schnelle Abschätzung der Güte einer Folge von Zufallszahlen benutzt werden kann, besteht darin, die Zufallszahlen mit einem Packprogramm zu komprimieren. Der Grad der möglichen Komprimierung korrespondiert nach Shannon umgekehrt proportional mit der Zufälligkeit der erzeugten Zahlen.

Ein wesentlich aussagekräftigerer Test ist aber der sehr bekannte χ^2-Test. Er prüft zwar das Gleiche wie der anfangs beschriebene grafische Test der Gleichverteilung, doch ist er wesentlich genauer, da er mit einem Rechenverfahren durchgeführt wird [Bronstein 96]. Ausgehend von der Annahme der Gleichverteilung der Zufallszahlen, kann man Mittelwert und Standardabweichung ermitteln. Auf der Grundlage einer χ^2-Verteilung lässt sich dann die Abweichung zur Normalverteilung errechnen. Daraus kann man eine numerische Aussage über die Verteilung der Zufallszahlen treffen.

Jedoch kann auch dieser Test nicht dazu benutzt werden, Aussagen über die Reihenfolge der auftretenden Zufallszahlen zu machen. Um dies zu überprüfen, greift man auf weitere statistische Tests zurück [Knuth 97], die z. B. Abstände von in den Zufallszahlen auftretenden Mustern auswerten (*Serial Test*). Analog dazu können die Abstände des Nichtauftretens von Mustern ermittelt werden (*Gap Test*). Ebenfalls sollten zur Beurteilung von Zufallszahlen die χ^2-Verteilung von vorkommenden Mustern (Poker Test) oder die χ^2-Verteilung von nicht vorkommenden Mustern (*Coupon Collector Test*) erfolgen.

Von einer gewissen Relevanz ist auch der Spektraltest (*Spectral Test*), bei dem der nachfolgende Wert einer Zufallszahl ermittelt wird [Knuth 97]. Bei der 2-dimensionalen Varian-

te dieses Test wird Zufallszahl und Nachfolger in einem xy-Koordinatensystem aufgetragen. Die 3-dimensionale Version erfordert den Nachfolger des Nachfolgers der Zufallszahl und noch eine z-Achse. Analog können auch n-dimensionale Spektraltests erstellt werden, wobei man dann aus einsichtigen Gründen auf eine grafische Darstellung verzichten muss.

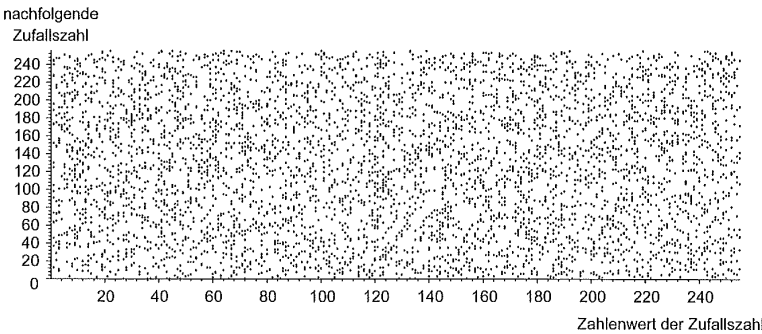


Bild 4.46 Grafische Darstellung der Verteilung des jeweiligen Nachfolgers von 5 000 Zufallszahlen mit einer Länge von einem Byte. Dies entspricht dem Spektraltest. Man sieht überblickshaft anhand des regelmäßigen Musters, dass der Wert der einer Zufallszahl nachfolgenden Zahl annähernd gleich verteilt ist. Die Zahlen wurden mit einem typischen Pseudozufallszahlengenerator einer Chipkarte erzeugt.

Um eine gesicherte und aussagekräftige Beurteilung eines Zufallszahlengenerators zu erstellen, müssen zumindest die oben beschriebenen Tests durchgeführt und ausgewertet werden. Zusätzliche Berechnungen und Tests können die erhaltenen Werte noch unterstützen. Erst damit lässt sich eine einigermaßen korrekte Aussage über die Qualität von Zufallszahlen erstellen.

Tabelle 4.23 Überblick und Zusammenfassung von üblichen statistischen Tests für Zufallszahlen.

Test und Referenz	Bemerkung
Coupon Collector Test [Knuth 97] Poker Test [Menezes 97]	χ^2-Verteilung des Nichtauftretens von Mustern in einer Zufallszahlenreihenfolge.
Frequency Test [Knuth 97, Menezes 97]	Zählen der Einsen in einer Zufallszahlenfolge.
Gap Test [Knuth 97]	Ermittlung der in einer Folge von Zufallszahlen nicht auftretenden Muster.
Long Run Test nach FIPS 140-2	Ermittlung, ob eine Folge von Nullen oder Einsen der Länge 34 Bit in einer 20000 Bit langen Zufallszahlenfolge vorkommt.
Monobit Test nach FIPS 140-2	Zählen der Einsen in einer 20000 Bit langen Zufallszahlenfolge.
Poker Test [Knuth 97]	χ^2-Verteilung des Auftretens von Mustern in einer Zufallszahlenreihenfolge.
Poker Test nach FIPS 140-1 Serial Test [Menezes 97]	Zählen von 4-Bit-Mustern in einer 20000 Bit langen Zufallszahlenfolge.
Runs Test nach FIPS 140-1	Ermitteln der maximalen Länge einer Folge von Nullen oder Einsen in einer 20000 Bit langen Zufallszahlenfolge.
Serial Test [Knuth 97]	Ermittlung der in einer Folge von Zufallszahlen auftretenden Muster.
Spectral Test [Knuth 97]	Ermittlung der Verteilung der Nachfolgewerte von Zufallszahlen.

Allerdings ist ein zu großer Aufwand für den Zufallszahlengenerator einer Chipkarte meist nicht gerechtfertigt, wenn man die Einsatzgebiete der Zufallszahlen betrachtet. Können etwa bei einer Authentisierung die Zufallszahlen vorausgesagt werden, dann wäre der Einfluss auf die Sicherheit eher gering. Ohne den geheimen Schlüssel zur Verschlüsselung der Zufallszahl ist ein Angriff nicht möglich.

Wesentlich problematischer wäre es aber, wenn der Zufallszahlengenerator manipuliert werden könnte, sodass er beispielsweise immer die gleichen Zufallszahlen erzeugt. Angriffe durch Wiedereinspielung wären dann nicht nur möglich, sondern auch erfolgreich. Dies wäre auch dann der Fall, wenn die Periode der Zufallszahlen sehr klein ist. Es ist also im Einzelfall genau abzuwägen, welche Haupteigenschaften die Zufallszahlen haben müssen, da dies natürlich auch Auswirkungen auf den Zufallszahlengenerator hat. Übertriebener Aufwand in diesem Bereich mag zwar zu einer sehr hohen Güte von Zufallszahlen führen, doch verursacht dies meist auch einen erhöhten Verbrauch an Speicherplatz, der gerade bei Chipkarten sehr begrenzt ist.

4.11 Authentisierung

Die Begriffe „Authentisierung" und „Authentifizierung" werden oft parallel benutzt, beschreiben jedoch das Gleiche. Der Hauptunterschied liegt nur in der Schwierigkeit der Aussprache des einen und in der nicht ganz korrekten wissenschaftlichen Bezeichnung des anderen. Im Folgenden wird hier als Zugeständnis der einfacheren Sprache der Erste der beiden Begriffe verwendet.

Zweck einer Authentisierung ist die Überprüfung der Identität und Authentizität eines Kommunikationspartners. Übertragen auf die Chipkartenwelt, bedeutet dies, dass die Karte oder das Terminal feststellt, ob der Kommunikationspartner ein echtes Terminal bzw. eine echte Karte ist. Im Zusammenhang mit der Authentitätsprüfung von Personen wird in diesem Buch aus Eindeutigkeitsgründen durchgehend der Terminus „Identifizierung" benutzt, obwohl dies im Prinzip auch unter den globalen Begriff „Authentisierung" fallen würde.

Die beiden Kommunikationsteilnehmer müssen zur Authentisierung ein gemeinsames Geheimnis besitzen, das mithilfe eines Authentisierungsverfahrens überprüft wird. Dies ist wesentlich sicherer als ein reines Identifizierungsverfahren, wie es z. B. die PIN-Überprüfung darstellt. Dort sendet man lediglich ein Geheimnis (die PIN) zur Karte, und diese bestätigt im Gutfall die Echtheit. Der Nachteil dieses Verfahrens ist, dass das Geheimnis im Klartext zur Karte gesendet wird, sodass ein Angreifer durch Abhören sehr einfach das Geheimnis, d. h. die PIN, erfahren kann.

Bei Authentisierungsverfahren ist es hingegen nicht möglich, durch Abhören der Leitung das gemeinsame Geheimnis herauszufinden. Es muss also nicht öffentlich über die Schnittstelle gesendet werden. Unterschieden wird auch noch zwischen statischer und dynamischer Authentisierung. Bei den statischen Verfahren werden immer die gleichen statischen Daten zur Authentisierung benutzt. Die dynamischen Verfahren hingegen sind so aufgebaut, dass sie vor einem Angriff durch Wiedereinspielen von in früheren Sitzungen aufgezeichneten Daten geschützt sind, da sie für jede Authentisierung eine unterschiedliche Datengrundlage verwenden.

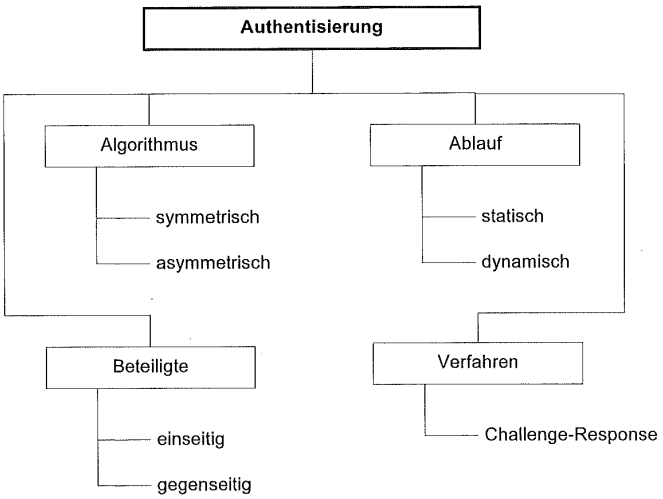


Bild 4.47 Klassifizierungsbaum der Authentisierung und Authentisierungsverfahren, die im Umfeld von Chipkarten benutzt werden.

Bei der Authentisierung unterscheidet man grundsätzlich noch zwischen einseitigen und gegenseitigen Verfahren. Die einseitige Authentisierung führt im Gutfall dazu, dass die Authentizität eines der beiden Kommunikationspartner sichergestellt ist. Bei der gegenseitigen Authentisierung sind am Ende des Verfahrens im Gutfall beide Partner authentisch.

Die auf einem kryptografischen Algorithmus basierenden und bei Chipkarten vorkommenden Authentisierungsverfahren teilt man noch in symmetrische und asymmetrische – Verfahren ein. Im Bereich der Chipkarten werden momentan fast ausschließlich symmetrische Verfahren eingesetzt. Die asymmetrischen, also auf RSA oder ähnlichen Algorithmen aufbauenden Verfahren, haben bei Chipkarten aufgrund der langsamen Ausführungsgeschwindigkeit zurzeit noch keine große praktische Bedeutung. Es ist aber absehbar, dass sich dies in Zukunft ändern wird. Im Prinzip funktionieren sie aber analog den symmetrischen Authentisierungsverfahren.

Zur Authentisierung von Geräten existieren mehrere Normen. Vor allem die ISO/IEC 9798 kommt hier in Betracht, in der in Teil 2 symmetrische und in Teil 3 asymmetrische Verfahren beschrieben sind. Grundsätzlich ist die fünfteilige Normenreihe ISO/IEC 9798 eine hervorragende Zusammenstellung der gängigen symmetrischen, asymmetrischen, MAC- und Zero-Knowledge-basierten Authentisierungsverfahren.

Das Prinzip der Authentisierung im Chipkartenbereich basiert immer auf dem Challenge-Response-Verfahren. Dabei stellt der eine Kommunikationspartner dem Anderen eine zufällig erzeugte Frage (*challenge*), dieser berechnet mit einem Algorithmus eine Antwort und sendet sie an den Fragesteller zurück (*response*). Der Algorithmus ist natürlich vorzugsweise eine Verschlüsselung mit einem geheimen Schlüssel, der das gemeinsame Geheimnis der beiden Kommunikationspartner darstellt.

4.11.1 Einseitige symmetrische Authentisierung

Mit einer einseitigen Authentisierung vergewissert man sich der Vertrauenswürdigkeit eines Kommunikationspartners. Dazu ist es notwendig, dass beide ein gemeinsames Geheimnis besitzen, dessen Kenntnis durch die Authentisierung überprüft wird. Das Geheimnis ist ein Schlüssel für einen Verschlüsselungsalgorithmus, an dem die gesamte Sicherheit des Authentifikationsverfahrens hängt. Würde dieser Schlüssel bekannt, könnte sich ein Angreifer genauso authentisieren wie der echte Kommunikationspartner.

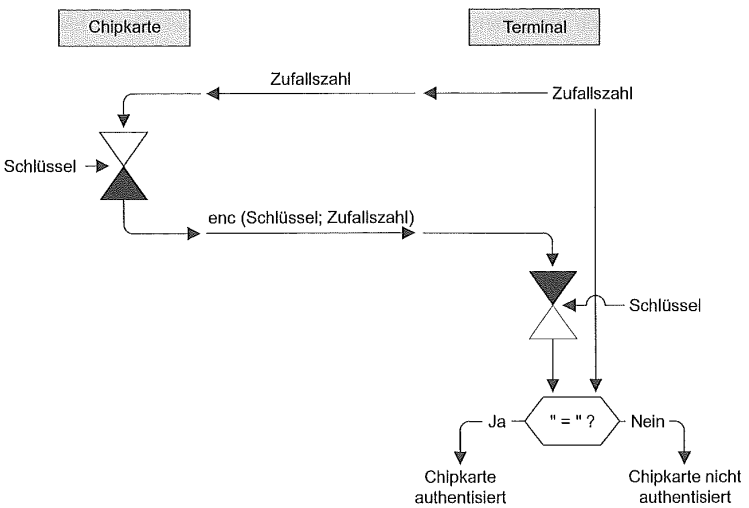


Bild 4.48 Das Prinzip der einseitigen Authentisierung mit einem symmetrischen Kryptoalgorithmus. Das Beispiel zeigt die Authentisierung der Chipkarte durch das Terminal, wie sie durch das Kommando INTERNAL AUTHENTICATE nach ISO/IEC 7816-4 realisiert werden kann.

Das Prinzip der einseitigen Authentisierung mit symmetrischen Kryptoalgorithmen lässt sich wie in der obigen Grafik darstellen. Dabei wurde der Übersichtlichkeit halber davon ausgegangen, dass ein Terminal eine Chipkarte authentisiert. Das Terminal stellt damit also fest, ob die Chipkarte vertrauenswürdig ist.

Das Terminal generiert eine Zufallszahl und sendet diese zur Chipkarte, was man als Challenge (Anfrage) bezeichnet. Nach dem Empfang dieser Zufallszahl wird sie von der Chipkarte verschlüsselt. Der dabei verwendete Schlüssel ist nur dem Terminal und der Chipkarte bekannt. Die Sicherheit des Verfahrens hängt von diesem Schlüssel ab, da nur der Besitzer des geheimen Schlüssels in der Lage ist, die richtige Antwort für das Terminal zu erzeugen.

Das Ergebnis der Verschlüsselung sendet die Chipkarte zum Terminal zurück. Dies ist die Antwort (*response*) auf die Anfrage (*challenge*). Das Terminal führt nun mit der erhaltenen und verschlüsselten Zufallszahl eine Entschlüsselung mit dem geheimen Schlüssel aus. Danach vergleicht sie das Ergebnis mit der ursprünglich an die Chipkarte

gesendeten Zufallszahl. Stimmt beides überein, so weiß das Terminal, dass die Chipkarte den geheimen Schlüssel kennt und schließt daraus, dass die Chipkarte authentisch ist.

Ein Angriff durch Wiedereinspielung einer vorher abgehörten Challenge oder Response ist nicht möglich, da in jeder Sitzung eine andere Zufallszahl erzeugt wird. Der einzige Angriff mit einigermaßen guten Erfolgsaussichten bestünde in einer systematischen Suche nach dem geheimen Schlüssel. Da Challenge und Response nichts anderes als Klartext-Schlüsseltext-Paare sind, könnte man durch einen Brute-force-Angriff den geheimen Schlüssel berechnen.

Besäßen alle Karten einer Anwendung den gleichen Schlüssel und würde dieser bekannt, so wäre das gesamte System diskreditiert. Um genau diesen Fall zu verhindern, werden in der Praxis grundsätzlich nur kartenindividuelle Schlüssel verwendet. Damit hat jede Karte einen individuellen Schlüssel, der aus einem nicht geheimen Kartenmerkmal abgeleitet werden kann. Als individuelles Merkmal kann die bei der Halbleiterproduktion in den Chip geschriebene Seriennummer oder eine andere Nummer, die bei jeder Chipkate anders ist, dienen.

Um nun den kartenindividuellen Schlüssel zu berechnen, fordert das Terminal von der Chipkarte die Chipnummer an. Diese Nummer ist individuell und auch einmalig im System, sodass es keine zweite gleiche Chipkarte gibt.

Der kartenindividuelle und geheime Authentisierungsschlüssel ist eine Funktion der Kartennummer und des Hauptschlüssels, der nur dem Terminal bekannt ist. In der Praxis verschlüsselt man einen Teil der Kartennummer mit dem Hauptschlüssel, und das Ergebnis daraus ist der kartenindividuelle Authentisierungsschlüssel. Dabei kann man einen DES- oder Triple-DES-Algorithmus verwenden.

Eines muss jedoch bedacht werden: Eine Kompromittierung des nur dem Terminal bekannten Hauptschlüssels würde das gesamte System kompromittieren, da mit dem Hauptschlüssel alle kartenindividuellen Authentisierungsschlüssel berechnet werden können. Deshalb muss der Hauptschlüssel im Terminal geschützt abgelegt sein (z. B. in einem Sicherheitsmodul) und, wenn möglich, im Angriffsfall aktiv gelöscht werden können.

Nachdem also das Terminal den für diese Karte notwendigen Authentisierungsschlüssel errechnet hat, folgt der übliche Ablauf im Rahmen des Challenge-Response-Verfahrens. Die Chipkarte erhält vom Terminal eine Zufallszahl, verschlüsselt diese mit ihrem kartenindividuellen Schlüssel und sendet sie an das Terminal zurück. Dieses führt analog der Chipkarte die Umkehrfunktion der Berechnung aus und vergleicht die beiden Ergebnisse. Stimmen sie überein, so besitzen sowohl Chipkarte als auch Terminal ein gemeinsames Geheimnis, nämlich den geheimen kartenindividuellen Schlüssel, und die Chipkarte ist vom Terminal authentisiert.

Der Vorgang der Authentisierung ist durch den Aufruf von DES-Algorithmen, sofern diese in Software realisiert sind und die Datenübertragung von und zur Karte etwas zeitintensiv. Dies kann bei manchen Anwendungen zu Problemen führen.

Unter den folgenden Annahmen kann man den Zeitbedarf für eine einseitige Authentisierung überschlagsmäßig errechnen. Angenommen sei eine Chipkarte mit einem Takt von 3,5 MHz, dem Übertragungsprotokoll T = 1, einem Teiler von 372 und einem DES-Algorithmus, der 17 ms für einen Block benötigt. Alle internen Routinen der Chipkarte seien hier ohne genauere Angabe mit 9 ms angenommen, was die Berechnung vereinfacht, das Endergebnis aber nur unwesentlich verfälscht.

Tabelle 4.24 Berechnung des Zeitbedarfs in der Chipkarte für eine einseitige Authentisierung unter Berücksichtigung der Übertragungszeit

Kommando	Zeitbedarf für Übertragung		Zeitbedarf für Berechnung	
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

Man sieht deutlich anhand der Berechnung, dass eine einzige Authentisierung ca. 65 ms benötigt. Dies kann im Normalfall in einer Anwendung ohne zeitliche Probleme ausgeführt werden.

4.11.2 Gegenseitige symmetrische Authentisierung

Das Prinzip der gegenseitigen Authentisierung (*mutual authentication*) beruht auf einer zweifachen einseitigen Authentisierung. Man könnte auch zwei einseitige Authentisierungen abwechselnd für beide Kommunikationspartner ausführen. Dies wäre dann im Prinzip eine gegenseitige Authentisierung. Da jedoch der Kommunikationsaufwand aus zeitlichen Gründen so gering wie möglich gehalten werden muss, gibt es ein Verfahren, in dem die beiden einseitigen Authentisierungen miteinander verflochten sind. Dabei erzielt man auch noch eine höhere Sicherheit als mit zwei nacheinander ausgeführten Authentisierungen, da es für einen Angreifer viel schwieriger ist, in den Kommunikationsablauf einzugreifen.

Damit das Terminal mit dem Hauptschlüssel aus der Kartennummer den kartenindividuellen Authentisierungsschlüssel berechnen kann, benötigt es als Erstes die Kartennummer. Nachdem das Terminal die Kartennummer erhalten hat, berechnet es den individuellen Authentisierungsschlüssel für diese Chipkarte. Dann fordert es von der Chipkarte eine Zufallszahl an und generiert selbst ebenfalls eine Zufallszahl. Nun setzt das Terminal beide Zufallszahlen vertauscht hintereinander, verschlüsselt sie mit dem geheimen Authentisierungsschlüssel und sendet den erhaltenen Schlüsseltext zur Karte. Das Vertauschen hat den Zweck, Challenge und Response unterschiedlich zu machen.

Diese kann den erhaltenen Block entschlüsseln und prüfen, ob die zuvor an das Terminal gesendete Zufallszahl mit der zurückerhaltenen übereinstimmt. Ist dies der Fall, weiß die Chipkarte, dass das Terminal den geheimen Schlüssel besitzt. Damit ist das Terminal gegenüber der Chipkarte authentisiert. Daraufhin vertauscht die Chipkarte die beiden Zufallszahlen, verschlüsselt sie mit dem geheimen Schlüssel und schickt das Ergebnis zum Terminal.

Das Terminal entschlüsselt den erhaltenen Block und vergleicht die zuvor an die Chipkarte gesendete Zufallszahl mit der erhaltenen. Stimmt diese mit der vormals gesendeten überein, so ist auch die Chipkarte gegenüber dem Terminal authentisiert. Damit ist die gegenseitige Authentisierung abgeschlossen, und sowohl Chipkarte als auch Terminal wissen, dass der jeweils andere vertrauenswürdig ist.

Um den Zeitbedarf der Kommunikation zu minimieren, kann die Chipkarte zusätzlich zur Kartennummer auch noch die Zufallszahl zurücksenden. Dies ist dann von Interesse, wenn die gegenseitige Authentisierung zwischen Chipkarte und einem Hintergrundsystem stattfindet. Die Chipkarte wird dabei direkt vom Hintergrundsystem transparent zum Terminal angesprochen. Die Datenübertragungsgeschwindigkeit ist dabei oft sehr niedrig, und der Kommunikationsablauf muss dadurch so stark wie möglich vereinfacht werden.

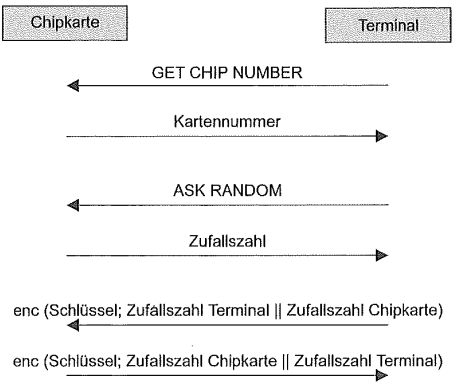


Bild 4.49 Die gegenseitige Authentisierung mit kartenindividuellem Schlüssel und einem symmetrischen Kryptoalgorithmus. Der Ablauf entspricht einer gegenseitigen Authentisierung von Chipkarte und Terminal, wie sie durch das Kommando MUTUAL AUTHENTICATE nach ISO/IEC 7816-8 realisiert werden kann.

Um den erheblichen Zeitaufwand, auch im Gegensatz zur einseitigen Authentisierung, aufzuzeigen, ist im Folgenden nochmals eine rechnerische Betrachtung aufgeführt. Die zugrunde liegenden Annahmen sind dabei analog der einseitigen Authentisierung. Man sieht, dass die gegenseitige Authentisierung beinahe dreimal so lange dauert als eine einseitige.

Tabelle 4.25 Berechnung des Zeitbedarfs in der Chipkarte für eine gegenseitige Authentisierung unter Berücksichtigung der Übertragungszeit. Es wurde angenommen, dass keine abgeleiteten Schlüssel Verwendung finden (GET CHIP NUMBER ist deshalb nicht notwendig) und sowohl die Erzeugung der Zufallszahl als auch die DES-Berechnung in Software auf der Chipkarte realisiert sind.

Kommando	**Zeitbedarf für Übertragung**		**Zeitbedarf für Berechnung**	
ASK RANDOM	28,75 ms		26 ms	
MUTUAL AUTHENTICATE	68,75 ms		95 ms	
	97,50 ms	+	121 ms	= 218,50 ms

Tabelle 4.26 Berechnung des Zeitbedarfs in der Chipkarte für eine gegenseitige Authentisierung unter Berücksichtigung der Übertragungszeit. Es wurde angenommen, dass keine abgeleiteten Schlüssel Verwendung finden (GET CHIP NUMBER ist deshalb nicht notwendig) und sowohl die Erzeugung der Zufallszahl als auch die DES-Berechnung in Hardware auf der Chipkarte realisiert sind.

Kommando	**Zeitbedarf für Übertragung**		**Zeitbedarf für Berechnung**	
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 Statische asymmetrische Authentisierung

Nur wenige Chipkarten-Mikrocontroller besitzen eine Recheneinheit, mit der RSA-Berechnungen durchgeführt werden können. Dies liegt vor allem daran, dass diese zusätzlichen Platz auf dem Chip benötigt, was den Preis erhöht.

Da nun aber ein zusätzliches asymmetrisches Authentisierungsverfahren vermehrten Schutz bedeutet, da ein Angreifer nicht nur einen kryptografischen Algorithmus brechen muss, sondern zwei, möchte man oft noch diese Art von Authentisierung verwenden. Um das Problem der nicht vorhandenen Recheneinheit auf der Chipkarte zu umgehen, fand man als Ausweg eine statische Authentisierung der Chipkarte durch das Terminal. Diese erfordert lediglich eine Verifizierung im Terminal. Eine zusätzliche Recheneinheit auf dem Terminal erhöht aber die Kosten im Verhältnis zum Gesamtpreis dieser Geräte nur unwesentlich, deshalb ist dieser Weg wesentlich kostengünstiger als die Verwendung spezieller Chipkarten-Mikrocontroller. Zudem ist das Verfahren viel schneller, da nur eine asymmetrische Verschlüsselung notwendig ist, im Gegensatz zu zwei bei einer dynamischen asymmetrischen Authentisierung.

Man erkauft sich diesen Kompromiss jedoch durch eine verminderte Sicherheit des Authentisierungsverfahrens. Ein Schutz gegen Wiedereinspielung ist durch das statische Verfahren natürlich nicht gegeben. Deshalb benutzt man es auch nur als zusätzliche Überprüfung der Authentizität der Karte, die zuvor bereits mit einem dynamischen symmetrischen Verfahren überprüft worden ist.

Das Verfahren funktioniert in seinem grundlegenden Ablauf folgendermaßen: Bei der Personalisierung werden in jede Chipkarte kartenindividuelle Daten eingetragen. Dies sind beispielsweise eine Kartennummer, der Name des Kartenbesitzers und seine Adresse. Über diese Daten, die während der Lebensdauer der Karte nicht veränderbar sind, wird während der Personalisierung eine digitale Signatur mit einem geheimen Schlüssel gerechnet. Der Schlüssel wird im System global verwendet. Benutzt man nun diese Karte an einem Terminal, so liest dieses aus einer Datei auf der Karte die Signatur und die signierten Daten aus. Das Terminal besitzt den für alle Chipkarten gültigen öffentlichen Schlüssel und kann die gelesene Signatur verschlüsseln und das Ergebnis mit den gelesenen Daten vergleichen. Stimmen beide überein, so ist die Karte durch das Terminal authentisiert.

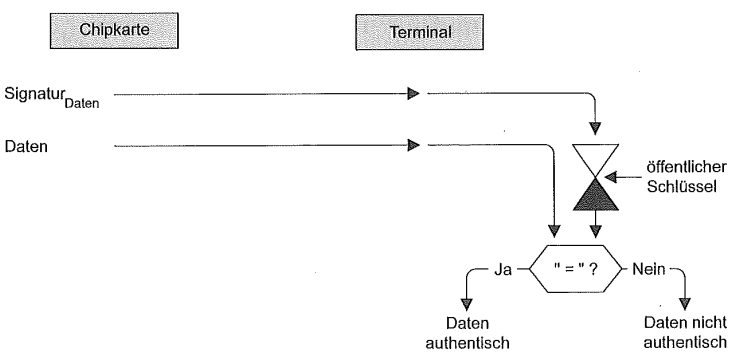


Bild 4.50 Das Prinzip der einseitigen, statischen und asymmetrischen Authentisierung einer Chipkarte durch das Terminal unter Verwendung von globalen Schlüsseln.

Das obige Verfahren hat neben dem fehlenden Schutz vor Wiedereinspielung einen weiteren Nachteil. Es werden für die Erstellung und Überprüfung der Signaturen globale Schlüssel benutzt. Der Schlüssel im Terminal erfordert zwar keinerlei Schutz, da er öffentlich ist, doch sollten in einem größeren System grundsätzlich keine für alle Karten einheitlichen Schlüssel verwendet werden. Kann ein solcher Schlüssel gebrochen werden, oder wird er aus anderweitigen Gründen bekannt, so ist diese Authentisierung im gesamten System wertlos. Es ist also notwendig, kartenindividuelle Schlüsselpaare für die statische Authentisierung einzuführen.

Dabei ergibt sich aber ein Problem mit der Speicherkapazität in den Terminals, da nun jedes Terminal alle vorhandenen öffentlichen Schlüssel zur Prüfung der Signatur kennen müsste. Selbst bei mittleren Systemen mit beispielsweise einer Million Chipkarten ergäbe dies bei 1024 Bit langen RSA-Schlüsseln einen notwendigen Speicherplatz zur Schlüsselablage von 128 MByte pro Terminal. Dies würde den Preis eines Terminal in Regionen treiben, die für einen Systembetreiber nicht mehr akzeptabel wären.

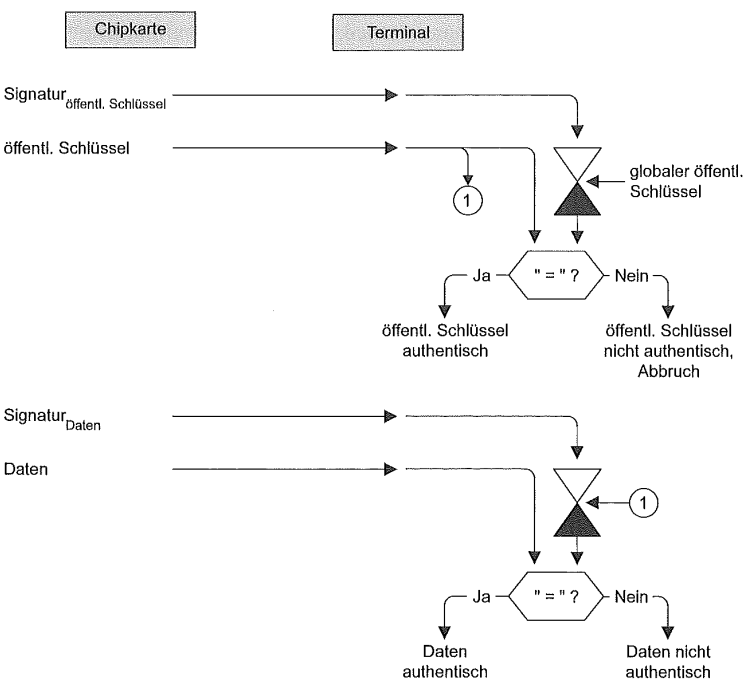


Bild 4.51 Das Prinzip der einseitigen, statischen und asymmetrischen Authentisierung einer Chipkarte durch das Terminal unter Verwendung von kartenindividuellen Schlüsseln.

Bei der Verwendung von symmetrischen Verfahren kann man auf recht einfache Weise die kartenindividuellen Schlüssel von einem Hauptschlüssel ableiten.[14] Bei asymmetrischen funktioniert dies aufgrund der Schlüsselerzeugung nicht. Deshalb geht man einen anderen Weg, wenn kartenindividuelle Schlüssel erforderlich sind. Man speichert den öffentlichen Schlüssel zur Überprüfung der Signatur mit dieser in der Karte. So sind zwar bei dem beschriebenen System mit einer Million Chipkarten noch immer 128 MByte Speicher zur Schlüsselablage erforderlich, doch sind diese in 128-Byte-Paketen auf einer Million Chipkarten verteilt. Das Terminal liest dann den öffentlichen Schlüssel aus einer Datei in der Chipkarte und kann mit ihm die Signatur prüfen. Damit umgeht man das Problem, dass alle öffentlichen Schlüssel eines Systems in jedem Terminal gespeichert werden müssen.

Nun könnte sich aber ein Angreifer ein Schlüsselpaar erzeugen und mit diesem die Daten in einer gefälschten Karte signieren. Das Terminal liest den öffentlichen Schlüssel und stellt anschließend fest, dass die Karte echt ist. Aus diesem Grund muss man obiges Verfahren noch etwas verfeinern. Dabei wird der in jeder Karte gespeicherte öffentliche und kartenindividuelle Schlüssel mit einem globalen geheimen Schlüssel unterschrieben und diese Unterschrift in der Karte gespeichert.
Das Terminal geht nun folgendermaßen vor: Zuerst liest es den öffentlichen und kartenindividuellen Schlüssel aus der Karte und prüft mit dem globalen öffentlichen Schlüssel die Authentizität des kartenindividuellen Schlüssels. Ist dieser authentisch, dann liest es erst die eigentlichen Daten und prüft sie durch den in der Chipkarte gespeicherten öffentlichen Schlüssel.

Diese beiden Verfahren finden bereits in manchen Systemen Einsatz und werden in den nächsten Jahren mit Sicherheit noch vermehrt benutzt. Sobald jedoch die Recheneinheiten für asymmetrische Kryptoalgorithmen keinen nennenswerten Preisunterschiede bei Chipkarten-Mikrocontrollern mehr verursachen, werden die beiden aufgeführten Authentisierungsverfahren stark an Bedeutung verlieren. Der größte Nachteil ist der fehlende Schutz vor Wiedereinspielung, der zwar mit manchen Tricks (Weiterverwendung der signierten Daten im nachfolgenden symmetrischen Kryptoalgorithmus) etwas entschärft werden kann, doch gleicht dies die Sicherheitsnachteile gegenüber dynamischen Authentisierungsverfahren nie ganz aus.

4.11.4 Dynamische asymmetrische Authentisierung

Die vorangehenden statischen asymmetrischen Verfahren hatten einige Nachteile. Diese kann man ausschalten, indem man die Authentisierung dynamisiert. Auf diese Weise schützt man sich vor der Wiedereinspielung von vorher abgehörten Daten. Das übliche Verfahren ist die Verwendung einer Zufallszahl, die als Eingangswert für einen kryptografischen Algorithmus dient. Allerdings benötigt man dafür in der Chipkarte eine Recheneinheit, die den asymmetrischen Kryptoalgorithmus ausführen kann.

Oben ist eine einseitige Authentisierung mit einem globalen öffentlichen Schlüssel aufgezeigt. Fordert man kartenindividuelle Authentisierungsschlüssel, so ist das unter 4.10.3 beschriebene Verfahren zur Speicherung und Authentisierung der kartenindividuellen öffentlichen Schlüssel notwendig.

[14] Siehe auch Abschnitt 4.8.1 „Abgeleitete Schlüssel".

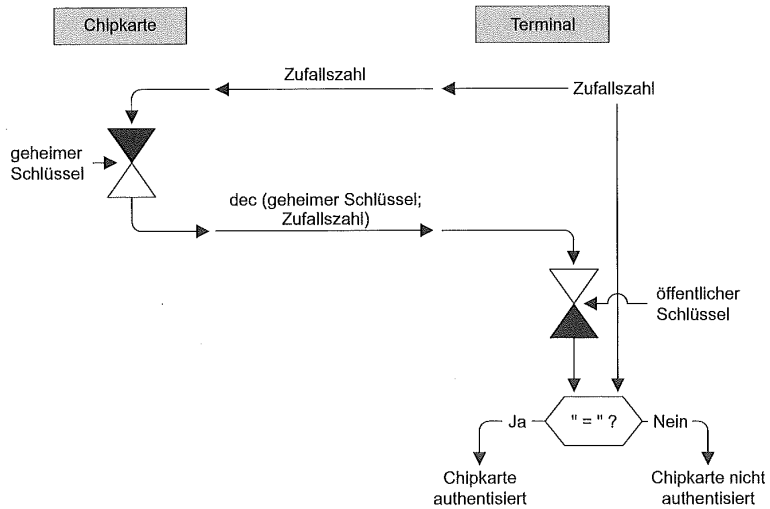


Bild 4.52 Das Prinzip der einseitigen, dynamischen und asymmetrischen Authentisierung einer Chipkarte durch das Terminal.

Analog der symmetrischen Authentisierung generiert das Terminal eine Zufallszahl und sendet diese zur Chipkarte. Diese entschlüsselt die Zufallszahl mit dem geheimen Schlüssel[15] und sendet das Ergebnis anschließend wieder zum Terminal. Dort befindet sich der globale öffentliche Schlüssel, welcher für die Verschlüsselung der empfangenen Zufallszahl benutzt wird. Ist das Ergebnis dieser Rechenoperation identisch mit der vorher zur Chipkarte gesendeten Zufallszahl, so ist die Chipkarte durch das Terminal authentisiert.

Eine gegenseitige Authentisierung von Chipkarte und Terminal ist in ihren Grundzügen analog der obig beschriebenen einseitigen aufgebaut. Diese erfordert aber aufgrund des hohen Datenübertragungsbedarfs und der aufwendigen asymmetrischen Verschlüsselungsverfahren relativ viel Zeit, sodass sie momentan sehr selten benutzt wird.

[15] Der Grund für eine Entschlüsselungsoperation bei der Erstellung einer Signatur liegt in der Konvention begründet, dass bei asymmetrischen Kryptoalgorithmen immer mit dem geheimen Schlüssel entschlüsselt und mit dem öffentlichen immer verschlüsselt wird. Die Konvention, dass der öffentliche Schlüssel für die Verschlüsselung verwendet wird, geht auf die Anfänge des RSA-Verfahrens zurück. Eine der damaligen Ideen war es, das RSA-Verfahren für Verschlüsselung von geheim zu haltenden Informationen durch Agenten in feindlichen Gebieten zu verwenden. Diese benötigen lediglich den RSA-Algorithmus und den öffentlichen Schlüssel, um damit ihre Nachrichten an die Agentenzentrale zu verschlüsseln. Die Entschlüsselung dieser Nachrichten mit dem geheimen Schlüssel würde dann in der auf freundlichem Gebiet befindlichen Agentenzentrale durchgeführt. Der größte Vorteil dieser Aufteilung liegt in der einfachen Schlüsselverteilung, da der Agent seinen Schlüssel im Prinzip ohne Sicherheitsmaßnahmen erhalten kann. Selbst wenn der Schlüssel bekannt würde, könnte damit niemand die verschlüsselten Nachrichten entschlüsseln, da er dazu den geheimen Schlüssel benötigt. Auf diese ersten stark militärisch geprägten Anwendungen des RSA-Verfahrens geht die noch immer gültige Konvention zurück, die darin besteht, mit dem öffentlichen Schlüssel zu verschlüsseln und mit dem geheimen Schlüssel wieder zu entschlüsseln.

4.12 Digitale Signatur

Digitale Signaturen, oft auch elektronische Unterschriften genannt, werden zur Feststellung der Authentizität von elektronisch übermittelten Nachrichten oder elektronischen Dokumenten verwendet. Durch Überprüfung der digitalen Signatur lässt sich feststellen, ob diese Nachrichten bzw. Dokumente verändert wurden.

Eine Unterschrift hat die Eigenschaft, dass sie nur von einem einzigen Menschen korrekt erzeugt, aber von allen Empfängern der Nachricht überprüft werden kann. Zumindest von denen, die die echte Unterschrift schon einmal gesehen haben, oder denen sie zum Vergleich vorliegt. Dies ist auch die wesentliche Eigenschaft einer digitalen Signatur. Nur eine Person bzw. eine Chipkarte kann ein Dokument unterschreiben, aber jedermann kann überprüfen, ob die Unterschrift echt ist. Aufgrund dieser geforderten Eigenschaft stellen asymmetrische kryptografische Verfahren die ideale Ausgangsbasis dar.

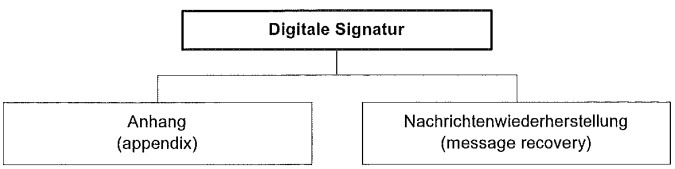


Bild 4.53 Klassifizierungsbaum der beiden grundsätzlichen Varianten „Daten zu signieren".

Die zu signierenden Nachrichten oder Dokumente sind im Regelfall mindestens einige tausend Bytes lang. Um nun die Rechenzeit zur Bildung der kryptografischen Prüfsumme in akzeptablen Grenzen zu halten, berechnet man diese Prüfsumme nicht über den gesamten Datenstring, sondern bildet zunächst einen Hash-Wert über den Datenstring. Hash-Funktionen[16] sind, vereinfacht ausgedrückt, Einwegfunktionen zur Komprimierung von Daten. Diese Komprimierung ist aber nicht umkehrbar, man kann also aus den komprimierten Daten nicht wieder das Original herstellen. Da die Berechnung eines Hash-Werts sehr schnell ist, sind sie die optimale Ergänzung zur Berechnung einer digitalen Signatur.

Der Begriff „digitale Signatur" wird üblicherweise nur in Verbindung mit asymmetrischen Kryptoalgorithmen benutzt, da sich diese aufgrund der Trennung in öffentliche und geheime Schlüssel sehr gut dafür eignen. In der Praxis finden aber trotzdem oft „Signaturen" Verwendung, die auf symmetrischen Kryptoalgorithmen beruhen. Mit diesen ist es aber nur möglich, ein Dokument auf seine Authentizität hin zu prüfen, wenn man den geheimen Schlüssel, der auch zur Generierung der Unterschrift verwendet wurde, besitzt. Dies ist im eigentlichen Sinne keine Signatur mehr, wird aber in der Praxis oft als solche bezeichnet. Zur Kennzeichnung des Verfahrens lässt man dabei jedoch das Wort „digital" weg.

Informationstechnisch gesehen kann eine digitale Signatur auf zweierlei Arten mit einer Nachricht verbunden werden: Die erste Art ist eine Variante einer kryptografischen Prüfsumme ähnlich einem MAC (*message authentication code*) über einen vorgegebenen Datenstring. Die Signatur wird dazu an die eigentliche Nachricht angehängt (*digital signature with appendix*). Dies hat den Vorteil, dass die Nachricht vollständig gelesen werden kann,

[16] Siehe auch Abschnitt 4.9 „Hash-Funktionen".

auch ohne vorher die Signatur prüfen zu müssen. Der Nachteil jedoch ist, dass die Nachricht um die Länge der digitalen Signatur vergrößert wird, was im Umfeld von Chipkarten durchaus Relevanz haben kann. Diesen Nachteil umgeht man mit der zweiten Art, eine digitale Signatur mit einer Nachricht zu verbinden (*digital signature with message recovery*). Dazu wird der eigentlichen Nachricht ein Hash-Wert über diese Nachricht nachgestellt und anschließend, beginnend vom Ende dieses Datenstrings her, der Eingangsblock für den Algorithmus der digitalen Signatur gebildet. Als Ergebnis verlängert sich die digital signierte Nachricht nur um die Länge des Hash-Wertes, doch kann sie nicht mehr vollständig gelesen werden, solange die digitale Signatur nicht geprüft ist.

Der Ablauf der Erzeugung einer digitalen Signatur mit Anhang lässt sich recht einfach darstellen. Aus der Nachricht, z. B. einer mit einer x-beliebigen Textverarbeitung erzeugten Datei, wird mit einem Hash-Algorithmus ein Hash-Wert gebildet. Dieser Hash-Wert wird mit einem asymmetrischen Kryptoalgorithmus, hier im Beispiel RSA, entschlüsselt. Das Ergebnis dieser Berechnung ist die eigentliche Unterschrift, die der Nachricht beigefügt wird.

Die Nachricht mit der Unterschrift kann nun über einen unsicheren Weg zum Empfänger geschickt werden. Dieser trennt Nachricht und Unterschrift wieder voneinander. Die Nachricht wird mit dem gleichen Hash-Algorithmus wie beim Sender komprimiert. Die digitale Signatur wird mit dem öffentlichen Schlüssel des RSA-Algorithmus verschlüsselt und mit dem Ergebnis der Hash-Berechnung verglichen. Sind beide Werte gleich, so wurde die Nachricht auf ihrem Übertragungsweg nicht verändert, im anderen Fall wurde entweder Nachricht oder Signatur während der Übertragung verändert. Die Authentizität ist damit nicht mehr gegeben, und der Inhalt der Nachricht kann nicht mehr als unverändert angenommen werden.

Die Aufgabe der Chipkarte in diesem Szenario gestaltet sich sehr einfach. Sie speichert zumindest den geheimen RSA-Schlüssel und entschlüsselt den Hash-Wert über die Nachricht, erstellt also damit die Signatur. Alles andere, wie Erzeugung des Hash-Werts oder das spätere Prüfen der Signatur, kann im Prinzip auch von einem PC erledigt werden.

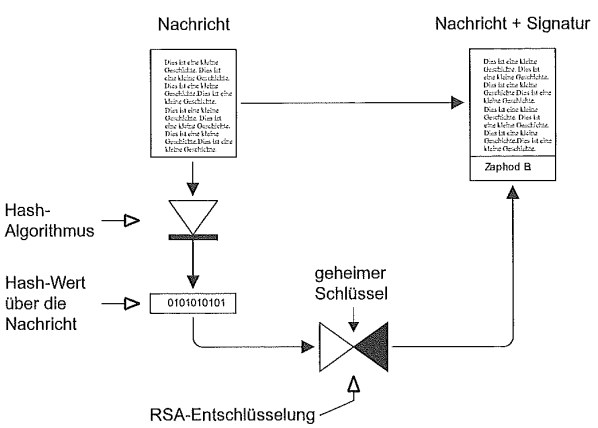


Bild 4.54 Signieren einer Nachricht mit dem RSA-Algorithmus durch Anhängen der erzeugten Signatur an die Nachricht (*digital signature with appendix*).

Optimal allerdings wäre es jedoch, wenn die Chipkarte die Nachricht über die Schnittstelle erhält, den Hash-Wert errechnet und dann dem Terminal signiert zurücksendet. Die Prüfung der Signatur könnte ebenfalls mit der Chipkarte durchgeführt werden. Dieses Verfahren ist zwar nicht sicherer als die bloße Berechnung der Signatur, doch wesentlich anwendungsfreundlicher. Hash-Algorithmen und RSA-Schlüssel können dann nämlich ohne Änderung von Programmen oder Daten auf dem PC durch bloßen Austausch der Chipkarte gewechselt werden.

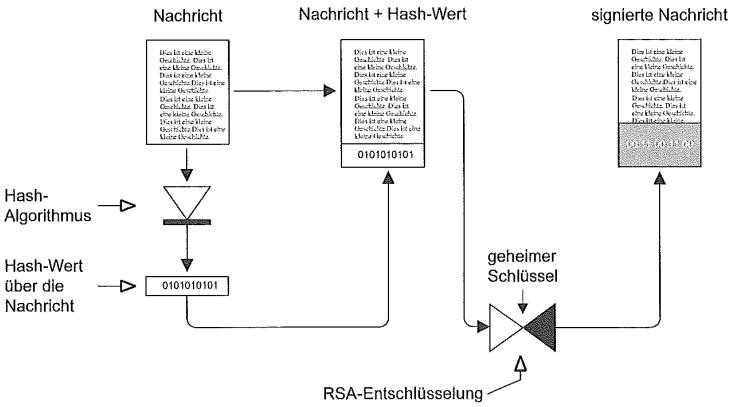


Bild 4.55 Signieren einer Nachricht mit dem RSA-Algorithmus durch Einbeziehung der Nachricht und eines Hash-Wertes über die Nachricht in die digitale Signatur (*digital signature with message recovery*).

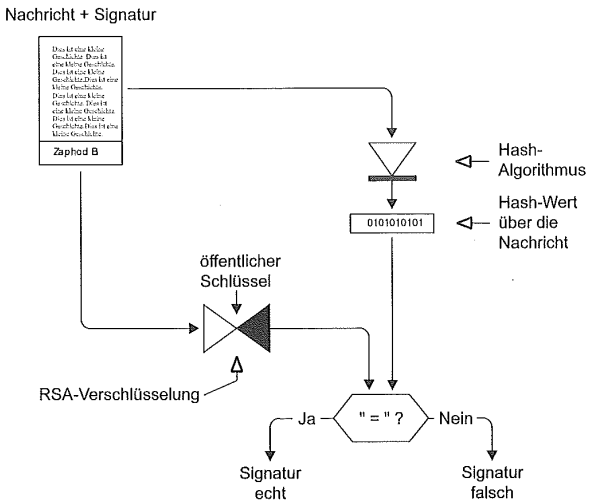


Bild 4.56 Prüfen einer mit dem RSA-Algorithmus signierten Nachricht, der die Signatur als Anhang beigefügt wurde (*digital signature with appendix*).

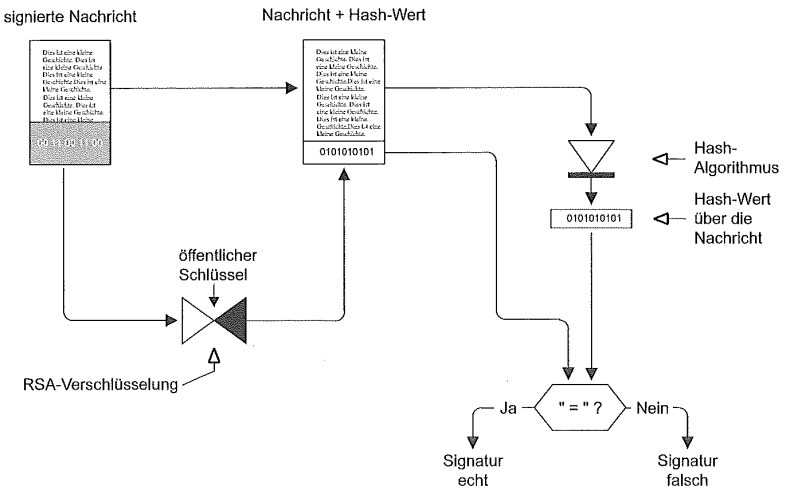


Bild 4.57 Prüfen einer mit dem RSA-Algorithmus signierten Nachricht, von der ein Teil für die Signatur verwendet wurde (*digital signature with message recovery*).

Die beiden Beispiele verwendeten zur Erzeugung und zur Prüfung der digitalen Unterschrift globale und damit für alle Chipkarten eines Systems gleiche Schlüssel. Soll dies aus Sicherheitsgründen geändert werden, sodass jede Chipkarte einen eigenen Schlüssel für die digitale Signatur hat, muss ein Verfahren wie in 4.10.3 beschrieben eingesetzt werden.

Zur Erstellung von digitalen Signaturen lässt sich nicht nur der RSA-Algorithmus verwenden, sondern es gibt auch ein speziell für dieses Einsatzgebiet entwickeltes kryptografisches Verfahren, nämlich der vom NIST (US National Institute of Standards and Technology) im Jahr 1991 vorgeschlagene DSA (*digital signature algorithm*), mit dem sowohl Signaturen erstellt als auch geprüft werden können. Im Gegensatz zum RSA-Algorithmus war der DSA-Algorithmus dahingehend ausgelegt, dass man damit keine Ver- und Entschlüsselungen durchführen konnte, was jedoch mittlerweile widerlegt wurde. Dies hatte gegenüber den strikten Exportbeschränkungen beim RSA-Algorithmus oder Algorithmen, die auf elliptischen Kurven beruhen, Vorteile für den internationalen Einsatz bzw. Export.

4.13 Zertifikate

Bei der Anwendung von digitalen Signaturen stößt man ziemlich schnell auf ein nicht zu unterschätzendes Problem. Derjenige, der die digitale Signatur einer Nachricht prüfen möchte, benötigt hierfür den passenden öffentlichen Schlüssel. Dieser kann aber nicht einfach ungeschützt verschickt werden, da der Empfänger sonst nicht prüfen kann, ob dieser Schlüssel authentisch ist. Der öffentliche Schlüssel muss also von einer vertrauenswürdigen Instanz unterschrieben werden, damit es möglich ist, seine Echtheit zu verifizieren. Diese signierende Instanz wird Zertifizierungsinstanz (*certification authority – CA*) genannt. Den von der Zertifizierungsinstanz unterschriebenen öffentlichen Schlüssel mit dazugehöriger digitaler Signatur und zusätzlichen Parametern bezeichnet man als Zertifikat (*certificate*).

In diesem Zusammenhang existiert eine weitere Instanz – das Trustcenter (TC). Ein Trustcenter erstellt und verwaltet Zertifikate, dazugehörige Sperrlisten und kann optional auch die Schlüsselgenerierung für digitale Signaturkarten vornehmen. In der Regel führt ein Trustcenter auch ein öffentliches Verzeichnis mit Zertifikaten, sodass der Prüfer einer signierten Nachricht den dazugehörigen signierten öffentlichen Schlüssel dort beispielsweise via Internet anfordern kann.

Ein Zertifikat enthält nun nicht nur den signierten öffentlichen Schlüssel, sondern auch noch eine große Zahl zusätzlicher Parameter und Optionen, da es möglich sein muss, ohne weitere Informationen den öffentlichen Schlüssel eines Zertifikates zu prüfen. Daraus folgt, dass beispielsweise der verwendete Signier- und Hash-Algorithmus eindeutig festge-

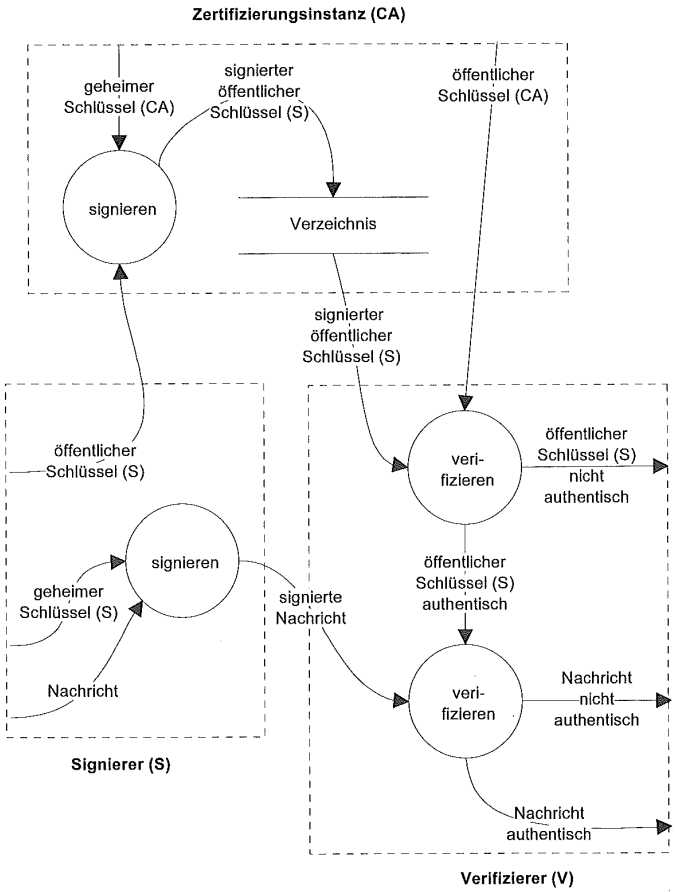


Bild 4.58 Datenflussdiagramm der grundlegenden Abläufe bei der Erzeugung und Prüfung einer übermittelten Nachricht unter Verwendung eines Zertifikats. Das Zertifikat wurde von einer Zertifizierungsinstanz erstellt und enthält den öffentlichen Schlüssel des Signierers und die Signatur der Zertifizierungsinstanz.

legt sein muss. Prinzipiell könnte nun jeder Signierende einen eigenen Aufbau von Zertifikaten festlegen. Allerdings wären diese Zertifikate dann nicht mehr austauschbar und hätten damit in der Regel ihren Sinn verfehlt, da gerade die Interoperabilität für Zertifikate eine charakteristische Eigenschaft zu sein hat.

Um diese Möglichkeit der Zusammenarbeit zu erreichen, existieren Normen, in denen die Struktur von Zertifikaten festgelegt ist. Die bekannteste diesbezügliche Norm ist X.509, die Aufbau und Codierung eines Zertifikats festlegt. Sie hat auch als ISO/IEC 9594-8 Eingang in die ISO/IEC-Normen gefunden.

Die umfangreiche Norm X.509 ist ein Rahmenwerk, in dem im Detail und in eindeutiger Schreibweise der Aufbau von Zertifikaten definiert ist. Sie ist die Grundlage vieler Anwendungen von digitalen Signaturen. Als Beispiel sei hier nur der Internetschutzmechanismus SSL (*secure socket layer*) und die Anwendungen PEM (*privacy enhanced mail*), SMIME (*secure multipurpose internet mail extensions*) und SET (*secure electronic transaction*) genannt.

Zur Beschreibung der Zertifikate wird in X.509 einheitlich ASN.1 benutzt, und für die eigentliche Codierung findet das weithin bekannte TLV-Schema nach DER (*distinguished encoding rules*) Anwendung[17]. Der mögliche Inhalt eines Zertifikates ist in der folgenden Tabelle im Überblick aufgezeigt. Eine kurze Einleitung und ein Überblick zum Thema X.509-Zertifikate findet sich in einem Papier von Peter Gutmann [Gutmann 98 b].

Tabelle 4.27 Ein typischer Aufbau eines Zertifikats nach X.509

Datenelement mit X.509 Bezeichnungen	**Erläuterung**
Version	Die Version von X.509, welche die Datenelemente dieses Zertifikats festlegt. In der Regel ist dies Version 3 von X.509.
Serial Number	Die Seriennummer des Zertifikats. Diese muss der Aussteller des Zertifikates vergeben, damit es eindeutig ist.
Signature Algorithm Identifier	Kennzeichen des verwendeten Kryptoalgorithmus für die digitale Signatur.
Issuer Name	Der Name des Ausstellers dieses Zertifikats. Der Name wird analog X.500 weltweit eindeutig beschrieben.
Validity Period	Der Gültigkeitszeitraum des Zertifikats.
Subject Name	Name der Instanz, dessen öffentlicher Schlüssel durch dieses Zertifikat als authentisch anerkannt werden soll. Der Name wird analog X.500 weltweit eindeutig beschrieben.
Public Key	Der öffentliche Schlüssel der Instanz, der mit diesem Zertifikat als authentisch anerkannt werden soll.
Signatur	Die digitale Signatur über die Daten des Zertifikats.

Es existieren bei X.509-Zertifikaten sehr viele optionale Datenfelder für die unterschiedlichsten Verwendungen. So ist es problemlos möglich, mehrere öffentliche Schlüssel in einem Zertifikat unterzubringen und diese auch von unterschiedlichen Zertifizierungsstellen unterschreiben zu lassen. Dies kann dazu führen, das ein Zertifikat mehrere Kilobyte Daten umfasst, was jedoch Probleme bereitet, wenn eine Chipkarte zur Prüfung des Zertifikats benutzt werden soll. Allerdings lassen sich mit diesem Mechanismus beispielsweise auch gegenseitige Zertifikate oder baumförmig organisierte Zertifikatshierarchien erzeugen. Ein typisches X.509-Zertifikat in einer Chipkarte hat normalerweise eine Größe von ca. 1 kByte.

[17] Siehe auch Abschnitt 4.1 „Strukturierung von Daten."

5 Chipkarten-Betriebssysteme

Es mutet vielleicht verwegen an, die wenigen tausend oder zehntausend Byte Programmcode für einen Chipkarten-Mikrocontroller als Betriebssystem zu bezeichnen, doch fällt dieses Programm tatsächlich voll und ganz unter diesen Begriff. Ein Betriebssystem nach der deutschen Norm DIN 44300 ist nicht mehr und nicht weniger als:

„Die Programme eines digitalen Rechensystems, die zusammen mit den Eigenschaften der Rechenanlage die Grundlage der möglichen Betriebsarten des digitalen Rechensystems bilden und insbesondere die Abwicklung von Programmen steuern und überwachen“.

Der Begriff Betriebssystem ist also nicht automatisch auf riesige Programme und Datenmengen beschränkt, sondern völlig unabhängig von Größen, da er ausschließlich die Funktionalität definiert. Es ist wichtig, dass man mit dem Begriff Betriebssystem nicht automatisch die mehrere Megabyte großen Programme für PCs oder Unix Computer assoziiert. Diese sind genauso spezifisch auf eine Mensch-Maschine-Schnittstelle über Monitor, Tastatur und Maus hin ausgelegt, wie die Chipkarten-Betriebssysteme auf die bidirektionale, serielle Schnittstelle zum Terminal hin gestaltet sind.

Letztendlich ist für ein Betriebssystem nur die Funktionalität entscheidend, die sich aus dem Zusammenwirken von zueinander passenden und aufeinander aufbauenden Biblio-

theksroutinen ergibt. Wichtig ist in diesem Zusammenhang auch, dass ein Betriebssystem eine Schnittstelle zwischen der Hardware des Rechners und der eigentlichen Anwendungssoftware ist. Dies hat für die Anwendungssoftware auch den großen Vorteil, dass diese nicht direkt auf die Hardware zugreifen muss und auf diese Weise zumindest eine, wenn auch oft sehr begrenzte, Portabilität erhalten bleibt.

Anfang der 90er-Jahre hat es nur sehr wenige echte Betriebssysteme für Chipkarten gegeben. Dies ist unter anderem auf die damals äußerst beschränkte Speicherkapazität der Chipkarten-Mikrocontroller zurückzuführen. Der Regelfall waren damals weniger Betriebssysteme als eine gut strukturierte Sammlung von Bibliotheksroutinen im ROM, die dann bei der Komplettierung je nach Anwendung aufgerufen wurden. Der Aufbau dieser Systeme war weitgehend monolithisch und ließ Änderungen nur unter großem Aufwand zu. Die nächste Generation wurde schon als geschichtetes Betriebssystem aufgebaut, bei dem es mit zahllosen Verfeinerungen bis heute geblieben ist.

Eines der ersten „richtigen" Chipkarten-Betriebssysteme war das von Giesecke und Devrient [GD] und der Gesellschaft für Mathematik und Datenverarbeitung (GMD) entwickelte STARCOS. Mit diesem Betriebssystem, dessen Entwicklung 1990 begann, war es schon damals möglich, mehrere Anwendungen unabhängig voneinander auf einer Chipkarte anzulegen, zu betreiben und zu verwalten.

Im Laufe der Entwicklung hat sich weltweit die Bezeichnung COS (*card operating system*) für Chipkarten-Betriebssysteme eingebürgert. Diese ist auch häufig als Teil des Produktnamens (z. B.: STARCOS, MPCOS) zu finden. Mittlerweile gibt es weltweit über ein Dutzend Hersteller von allgemeinen und anwendungsunabhängigen Chipkarten-Betriebssystemen, welche in der folgenden Tabelle im Überblick aufgezeigt sind.

Tabelle 5.1 Einige Beispiele von Chipkarten-Betriebssystemen unterschiedlicher Hersteller und deren WWW-Adressen. Diese Liste stellt lediglich eine Auswahl dar.

Name des Betriebssystems	Hersteller
GemXplore, GPK, MPCOS	Gemplus [Gemplus]
STARCOS, STARSIM, STARDC	Giesecke & Devrient [GD]
Multos	Maosco [Maosco]
AuthentIC, SIMphonic	Oberthur [Oberthur]
Micardo	Orga [Orga]
Cyberflex, Multiflex, Payflex	Schlumberger [Schlumberger]
CardOS	Siemens [Siemens]
TCOS	Telesec [Telesec]

Es ist vorstellbar, dass es mittelfristig zu einer Konsolidierung der mit unterschiedlichsten Eigenschaften und Funktionen ausgestatteten Chipkarten-Betriebssystemen kommen wird und damit zu einem ähnlichen Effekt wie bei den PCs, die mittlerweile alle mit einem „Einheitsbetriebssystem" betrieben werden. Ob dies auch die Zukunft bei den Chipkarten-Betriebssystemen sein wird, bleibt abzuwarten, denn hier sind die Rahmenbedingungen für eine Einheitslösung weniger günstig. Extreme Anforderungen im Bereich Sicherheit und Softwarequalität, Speicherplatzmangel und die Forderung nach Vertraulichkeit der Betriebssystemsoftware haben durchaus das Potenzial, ein universelles und alle möglichen Wünsche befriedigendes Chipkarten-Betriebssystem auf absehbare Zeit unmöglich zu machen.

Dieses Kapitel versucht anhand der in diversen Spezifikationen, Normen und Beschreibungen über Software für Chipkarten die möglichen Aspekte und Varianten heutiger Chipkarten-Betriebssysteme zu beleuchten.

5.1 Bisherige Entwicklung der Betriebssysteme

Die Evolution von Betriebssystemen im Chipkartenbereich hat die gleichen Phasen der Entwicklung durchlaufen wie bei alle anderen Computersystemen. Die anfänglichen Spezialprogramme für eine einzige Anwendung wurden immer weiter verallgemeinert und erweitert, sodass am Ende der Entwicklung ein einfach und strukturiert zu benutzendes Betriebssystem, das allgemein einsetzbar ist, entstand.

Die Programme für Chipkarten-Mikrocontroller in den Anfängen dieser Entwicklung um 1980 kann man aus heutiger Sicht eigentlich noch nicht als richtiges Betriebssystem bezeichnen. Dies war schlichtweg Anwendersoftware, die als ROM in einem Chip verankert war. Doch weil die Herstellung maskenprogrammierter Mikrocontroller teuer und zeitraubend ist, entstand sehr schnell das Bedürfnis nach allgemein einsetzbaren Kernroutinen, auf die dann im EEPROM bei Bedarf spezielle Anwendungssoftware aufbauen kann. Dadurch stieg aber auch der Speicherbedarf an, was bei einigen Firmen zu einer Gegenbewegung wieder hin zur Spezialsoftware für eine einzige Anwendung führte.

Da sich aber vom Markt her die Nachfrage nach individuell angepassten Lösungen bis heute laufend verstärkt, sind die Betriebssystem-Hersteller mehr oder minder gezwungen, ihre angebotenen Programme dahingehend auszulegen. Nur im Falle von Anwendungen mit sehr großen Stückzahlen geht man heute noch den individuellen Weg einer speziell entwickelten ROM-Software. Der Regelfall sind aber die allgemein einsetzbaren und auch auf genormten Kommandos basierenden Betriebssysteme, die von ihren Prinzipien her für jede Anwendung einsetzbar sind. Sollte dies aus speziellen Gründen einmal nicht möglich sein, dann sind sie zumindest so aufgebaut, dass sie mit geringem Aufwand und in kurzer Zeit an die Anforderungen jeder Anwendung angepasst werden können.

Die „historische" Entwicklung der Chipkarten-Betriebssysteme von 1980 bis heute lässt sich sehr gut anhand der Chipkarten für die Mobiltelefonnetze in Deutschland aufzeigen. Die im C-Netz, dem Vorgänger von GSM (D-Netz) in Deutschland, seit 1987 eingesetzte Chipkarte hat ein Betriebssystem, das für diese eine Anwendung hin optimiert wurde. Dazu gehören ein eigenes Übertragungsprotokoll, Spezialkommandos und ein auf die Anwendung zugeschnittener Dateiaufbau. Alles im allem hat die Karte sehr wohl ein vollständiges Betriebssystem, nur ist es komplett auf das Einsatzgebiet „C-Netz" zugeschnitten. Die auf dem Betriebssystem aufbauende Anwendung war in ihren wesentlichen Teilen auf die speziellen Anforderungen des bibliotheksorientierten Betriebssystems und der darunter liegenden Hardware zugeschnitten.

Der nächste Schritt war der Übergang von der Speziallösung hin zu einer etwas offeneren Architektur des Betriebssystems. Ein Vertreter davon sind die ersten GSM-Karten, die wesentlich offener und multifunktional aufgebaut sind. Als die GSM-Chipkarten spezifiziert wurden, gab es bereits Normenentwürfe für den Befehlssatz und die Datenstrukturen von

Chipkarten, sodass der Grundstein für eine Kompatibilität zwischen den einzelnen Betriebssystemen gelegt war. Auf dieser Basis wurde dann Schritt für Schritt weiterentwickelt. Die Anwendungen auf diesen monolithischen Betriebssystemen sind weitgehend unabhängig von der Hardware und bauen auf verschiedenen Schnittstellen des Betriebssystems auf.

Moderne Betriebssysteme für GSM weisen heute Funktionen wie Speicherverwaltung, mehrere Dateistrukturen und Zustandsautomaten auf, die sie sehr nahe an die Möglichkeiten der Multiapplication-Betriebssysteme heranbringen. Sie können mehrere Anwendungen unabhängig voneinander verwalten, ohne dass diese sich gegenseitig in irgendeiner Form beeinflussen. Auch besitzen sie meist sehr aufwendige Zustandsautomaten, eine große Menge an Kommandos und manchmal mehrere Übertragungsprotokolle.

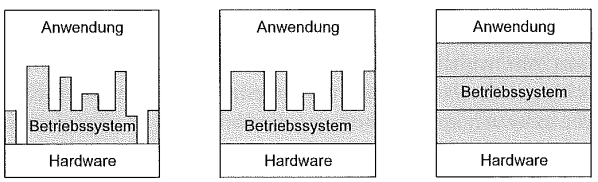


Bild 5.1 Schematische Darstellung der zeitlichen Entwicklung der Chipkarten-Betriebssysteme. Neue Betriebssysteme folgten bis ca. 1991 dem bibliotheksorientierten Ansatz und wurden bis ca. 1998 von monolithischen Betriebssystemen abgelöst, die ihrerseits mittlerweile durch die schichtenorientierten Betriebssysteme abgelöst wurden. Das Beispiel zeigt jeweils aus Vereinfachungsgründen jeweils nur eine Anwendung.

Doch auch diese Entwicklung setzt sich fort. Die Chipkarte in Mobiltelefonen übernimmt zunehmend Funktionen des Mobiltelefons selber, wie beispielsweise die Ansteuerung des Displays und die Abfrage der Tastatur. Um hier so flexibel wie möglich zu sein, muss mit einem bisher mit einem in der Chipkartenwelt eisern festgehaltenen Grundsatz gebrochen werden. Ein modernes Chipkarten-Betriebssystem muss fähig sein, von Dritten erstellten Programmcode auf der Chipkarte auszuführen. Dass dabei andere auf der Chipkarte befindliche Anwendungen weder in ihrer Funktion gestört noch in ihrer Sicherheit beeinträchtigt werden dürfen, ist bei einem Chipkarten-Betriebssystem eine Selbstverständlichkeit. Alle diese Betriebssysteme sind schichtenorientiert aufgebaut, sodass nur noch die unteren Schichten Abhängigkeiten von der Hardware aufweisen, die über die darauf aufbauenden Schichten immer weiter abstrahiert werden.

Die Entwicklung bei Chipkarten-Betriebssystemen könnte wohl in absehbarer Zukunft über mehrere Schritte zu einem internationalen Quasi-Standard für allgemein einsetzbare Betriebssysteme führen, so wie dies bei vielen anderen Betriebssystemen bereits heute der Fall ist. Nach einigen Jahren setzt sich über kurz oder lang immer ein so genannter Industriestandard durch, den alle Wettbewerber im Markt als kleinsten gemeinsamen Nenner unterstützen müssen, wenn sie weiterhin erfolgreich bleiben wollen. Dieser Standard existiert momentan in der Chipkartenwelt noch nicht, doch die ersten Anzeichen dafür sind absch-

bar. Grundlage bilden hier aber, anders als beispielsweise in der PC-Welt, internationale Normen und Spezifikationen. Dies sind hauptsächlich die ISO/IEC 7816-Normenreihe, die UICC-Spezifikationen (TS 102.221) und die EMV-Spezifikationen.

Tabelle 5.2 Typische Beispiele für den Umfang von Sourcecode und den charakteristischen Programmstrukturen bei Chipkarten-Betriebssystemen für die Telekommunikations-Anwendungen GSM. Würde das in der Tabelle aufgeführte Betriebssystem aus dem Jahr 2002 um die Funktionalität von Java Card ergänzt, so ergäbe dies einen Sourcecodeumfang von ca. 180000 Zeilen.

Art des Chipkarten-Betriebssystems	**Betriebssystem für GSM um 1990, implementiert in Assembler**	**Betriebssystem für GSM um 1997, implementiert in Assembler**	**Betriebssystem für GSM um 1996, implementiert in C**	**Betriebssystem für GSM um 2002, implementiert in C**
Funktionalität	GSM 11.11, Administrative Kommandos	GSM 11.11, OTA, Administrative Kommandos	GSM 11.11, Administrative Kommandos	196 kByte ROM 68 kByte EEPROM 4 kByte RAM
Mikrocontroller	6 kByte ROM 3 kByte EEPROM 128 Byte RAM	16 kByte ROM 8 kByte EEPROM 256 Byte RAM	16 kByte ROM 8 kByte EEPROM 256 Byte RAM	GSM 11.11, GSM 11.14, WIM, 3 · Mikrobrowser, OTA, Administrative Kommandos
Größe des gesamten gelinkten Objektcodes (ROM + EEPROM)	10 kByte + 0,3 kByte	16 kByte + 0,8 kByte	16 kByte + 0,8 kByte	190 kByte + 4,9 kByte
Anzahl der Zeilen Sourcecode inkl. Kommentare	≈10100	≈22000	≈12000	≈120000[1]
Anzahl der Dateien mit Sourcecode	22	14	37	184
Anzahl der Sprünge in Unterprogramme (bei Assembler) bzw. Anzahl der Funktionen (bei C)	≈470	≈930	≈115	≈900
Anzahl der Returns	≈95	≈35	≈205	≈1500
Anzahl der Konstanten	≈360	≈250	≈100	≈150
Anzahl der Verzweigungen (bei Assembler) bzw. IF-Anweisungen (bei C)	≈200	≈560	≈1100	≈3100

[1] Das im Jahr 2000 größte bekannte Softwareprogramm war Windows 2000 mit 29 Millionen Zeilen Sourcecode.

5.2 Grundlagen

Die Betriebssysteme für Chipkarten weisen im Gegensatz zu den allgemein bekannten Betriebssystemen keine Benutzeroberfläche und keine Zugriffsmöglichkeiten auf externe Speichermedien auf, da sie auf eine ganz andere Funktionalität hin optimiert sind. Die Sicherheit bei der Ausführung von Programmen und der geschützte Zugriff auf Daten hat dabei die oberste Priorität. Sie haben aufgrund der Einschränkungen durch den zur Verfügung stehenden Speicherplatz einen sehr kleinen Codeumfang, der im Bereich zwischen 3 und 250 kByte liegt. Die untere Grenze steht dabei für Spezialanwendungen und die obere für Multiapplication-Betriebssysteme mit z. T. Interpreter für nachladbaren Programmcode (z. B. Java Card). Der durchschnittliche Speicherbedarf liegt aber meist im Bereich bis 64 kByte bei Betriebssystemen ohne von dritten nachladbaren Programmcode.

Die Programmmodule sind als ROM-Code geschrieben, was dazu führt, dass die Methoden der Programmierung eingeschränkt sind, da viele bei RAM-Programmcode übliche Abläufe (z. B. selbstmodifizierender Code) nicht möglich sind. Der ROM-Code ist auch der Grund dafür, dass nach der Programmierung und Herstellung des ROMs auf dem Mikrocontroller keinerlei Änderungen mehr vorgenommen werden können. Die Beseitigung eines Fehlers ist deswegen extrem teuer und mit einer Durchlaufzeit von 10 bis 12 Wochen verbunden. Ist die Chipkarte beim Endbenutzer angelangt, lassen sich Fehler nur mehr durch groß angelegte Umtauschaktionen beseitigen, die den Ruf eines auf Chipkarten basierenden Systems ruinieren können. Eine „quick and dirty"-Programmierung verbietet sich deshalb von selbst. Der zeitliche Aufwand für Test und Qualitätssicherung ist deshalb im Regelfall deutlich höher als für die Programmierung.

Doch müssen diese Betriebssysteme neben der extremen Fehlerarmut auch sehr zuverlässig und robust sein. Sie dürfen durch kein von außen kommendes Kommando in ihrer Funktion und vor allem in ihrer Sicherheit beeinträchtigt werden. Systemzusammenbrüche oder unkontrollierte Reaktionen auf ein fehlerhaftes Kommando oder durch ausgefallene Seiten im EEPROM dürfen auf keinen Fall vorkommen.

Vom Standpunkt des Betriebssystemdesigns aus ist es leider so, dass die Realisation bestimmter Mechanismen von der verwendeten Hardware beeinflusst wird. Es ist vor allem der sichere Zustand des EEPROMs, der zwar nur einen kleinen, aber doch spürbaren Einfluss auf die Gestaltung des Betriebssystems hat. So müssen beispielsweise alle Fehlbedienungszähler so gestaltet sein, dass ihr höchster Wert mit dem gelöschten Zustand des EEPROMs zusammenfällt. Ist dies nicht der Fall, könnte man beispielsweise beim Schreiben des Fehlbedienungszählers durch gezieltes Abschalten der Spannungsversorgung diesen wieder auf seinen Ausgangswert zurücksetzen. Das ist deshalb möglich, da vor bestimmten Schreibzugriffen auf das EEPROM dieses gelöscht werden muss. Würde man nun genau im richtigen Augenblick nach dem Löschen und vor dem erneuten Schreiben die Spannungsversorgung abschalten, dann befände sich der Teil des EEPROMs für den Fehlbedienungszähler im gelöschten Zustand, und bei falscher Gestaltung des Betriebssystems wäre der Fehlbedienungszähler wieder auf seinem Ausgangswert gesetzt. Diesem Angriff kann entweder durch die vorgenannte richtige Codierung der Zählweise oder durch atomare Abläufe beim Schreiben des Fehlbedienungszählers begegnet werden. Ähnlich verhält es sich mit Fehlbedienungszähler und dem sicheren, d. h. energieärmsten Zustand des EEPROMs. Der

Fehlbedienungszähler muss so codiert sein, dass sein Maximalwert immer dem sicheren Zustand des EEPROMs entspricht. Ist dies nicht der Fall, könnte man beispielsweise durch punktuelles Erhitzen von EEPROM-Zellen den Fehlbedienungszähler wieder zurücksetzen. Dies sind nur zwei Beispiele für Hardware-Abhängigkeiten beim Aufbau eines Chipkarten-Betriebssystem, es gäbe noch viele weitere. Ein Chipkarten-Betriebssystem muss aus Sicherheitsgründen eng mit der Hardware des verwendeten Mikrocontrollers verknüpft werden und kann deshalb nicht vollständig Hardware-unabhängig gebildet werden.

Der Begriff „Sicherheitsbetriebssystem" enthält noch einen anderen Aspekt. Falltüren und andere Hintereingänge für Systemprogrammierer, wie sie bei großen Systemen immer wieder vorkommen und sogar durchaus üblich sind, müssen bei Chipkarten-Betriebssystemen gänzlich ausgeschlossen sein. Es darf z. B. keine Möglichkeit geben, am Betriebssystem vorbei mit irgendeinem Mechanismus Daten unautorisiert auszulesen.

Nicht zu unterschätzen ist außerdem die erforderliche Leistungsfähigkeit. Die im Betriebssystem vorhandenen kryptografischen Funktionen müssen innerhalb sehr kurzer Zeit ablaufen. So ist es üblich, während der Entwicklung in wochenlanger Kleinarbeit die entsprechenden Algorithmen in Assembler zu optimieren. Aufgrund der verwendeten Hardware-Plattformen und der geforderten Zuverlässigkeit ist es einleuchtend, dass Multitaskingfunktionalität in Chipkarten-Betriebssystemen keine Verwendung findet. Die Beschränkung auf eine einzeln ablaufende Task verhindert aber leider auch den Einsatz von Schutzprozessen, die Teile des Betriebssystems in Ablauf und Randbedingungen überwachen.

Zusammenfassend hat ein Chipkarten-Betriebssystem die folgenden Hauptaufgaben:

- Datenübertragung von und zur Chipkarte
- Ablaufsteuerung der Kommandos
- Dateiverwaltung
- Verwaltung und Ausführung von kryptografischen Algorithmen
- Verwaltung und Ausführung von Programmcode

Kommando-Abarbeitung

Die typische Kommando-Abarbeitung innerhalb des Chipkarten-Betriebssystems ohne nachladbaren Programmcode, das keinen nachladbaren Programmcode unterstützt, läuft wie folgt ab: Alle Kommandos an die Chipkarte empfängt diese über die serielle I/O-Schnittstelle. Fehlererkennungs- und -korrekturmechanismen führt der I/O-Manager bei Bedarf völlig unabhängig von den Übrigen, darauf aufbauenden Schichten aus. Nachdem ein Kommando vollständig und fehlerfrei empfangen wurde, muss der Secure Messaging Manager diesen gegebenenfalls entschlüsseln oder auf Integrität prüfen. Findet keine gesicherte Datenübertragung statt, ist dieser Manager sowohl für Kommando als auch Antwort völlig transparent.

Nach dieser Bearbeitung versucht die darüberliegende Schicht, der Kommandointerpreter, das Kommando zu decodieren. Ist dies nicht möglich, folgt ein Aufruf des Returncode-Managers, welcher einen entsprechenden Returncode generiert und via I/O-Manager an das Terminal zurücksendet. Es kann notwendig sein, den Returncode-Manager applikationsspezifisch zu gestalten, da die Returncodes nicht zwangsläufig für alle Anwendungen einheitlich sind. Konnte das Kommando jedoch decodiert werden, ermittelt der Logical

Channel Manager den angewählten Kanal, schaltet auf dessen Zustände um und ruft dann im Gutfall den Zustandsautomaten auf.

Dieser prüft nun, ob das Kommando an die Chipkarte mit den gesetzten Parametern im aktuellen Zustand überhaupt erlaubt ist. Ist das der Fall, wird der eigentliche Programmcode des Anwendungskommandos ausgeführt, welcher die Abarbeitung des Kommandos übernimmt. Falls das Kommando im aktuellen Zustand verboten ist oder die Parameter dazu nicht erlaubt sind, erhält das Terminal über Returncode Manager und I/O-Manager eine entsprechende Meldung.

Ist es notwendig, während der Kommandobearbeitung auf eine Datei zuzugreifen, geschieht dies nur über die Dateiverwaltung, die alle logischen Adressen in physikalische Adressen des Chips umsetzt. Weiterhin überwacht sie die Adressen hinsichtlich der Bereichsgrenzen und prüft die Zugriffsbedingungen auf die jeweilige Datei.

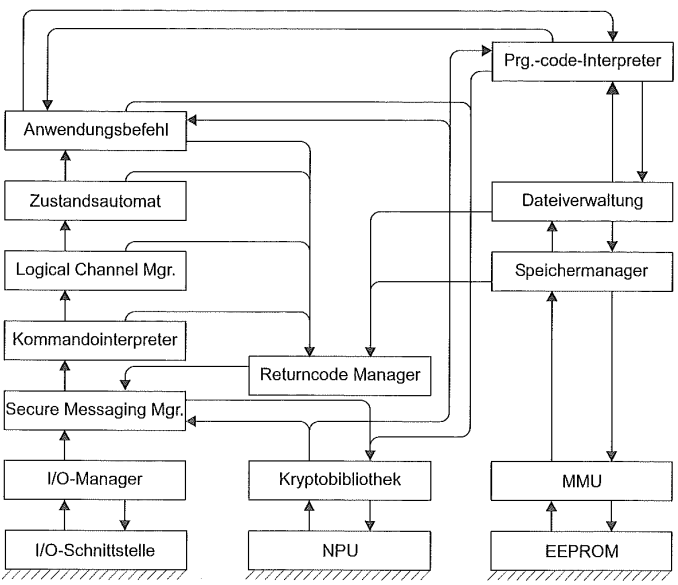


Bild 5.2 Ablauf der Kommando-Abarbeitung innerhalb eines Chipkarten-Betriebssystems. Die Hardware-Ferne nimmt in der Zeichnung nach oben hin zu. Der Programmcode-Interpreter ist optional.

Die Dateiverwaltung selber benutzt einen weiteren Speichermanager, der die komplette Verwaltung des physikalisch adressierten EEPROMs übernimmt. Damit ist sichergestellt, dass nur in diesem Programmmodul mit echten physikalischen Adressen gearbeitet wird, was die Portabilität und Sicherheit des ganzen Betriebssystems erheblich steigert.

Die Erzeugung der Antwortcodes übernimmt ein zentraler Returncode Manager, der dann für den aufrufenden Programmteil jeweils die komplette Antwort erzeugt. Diese Schicht übernimmt für alle anderen Teile des Betriebssystems die Verwaltung und Erzeugung aller verwendeten Returncodes.

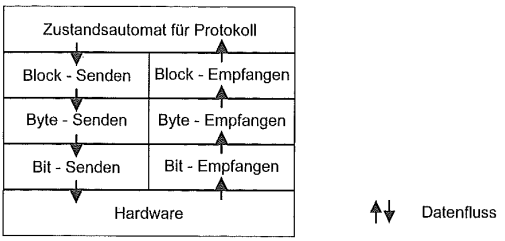


Bild 5.3 Teilablauf der Kommandoabarbeitung innerhalb des I/O-Managers in einem Chipkarten-Betriebssystem.

Da Chipkarten-Betriebssysteme meist kryptografische Funktionen nutzen, ist in der Regel noch eine eigene und vom Rest des Betriebssystems abgetrennte Bibliothek mit kryptografischen Funktionen vorhanden. Diese Kryptobibliothek dient allen anderen Modulen als zentrale Anlaufstelle bei Benutzung von Kryptofunktionen.

Zusätzlich zu diesen Schichten kann im Bereich oberhalb der Anwendungskommandos noch ein Interpreter oder ein Prüfprogramm für ausführbare Dateien vorhanden sein. Dieses überwacht die in den Dateien enthaltenen Programme, führt sie aus oder interpretiert sie. Der genaue Aufbau und die Implementation hängen davon ab, ob überhaupt Dateien mit ausführbarem Code vorgesehen sind und ob Maschinencode für den Prozessor oder zu interpretierender Code abgespeichert ist. Diese ganze Thematik ist ausführlich in Abschnitt 5.10 „Chipkarten-Betriebssysteme mit nachladbarem Programmcode" dargestellt.

Profile von Chipkarten

Anders als im Bereich der PC-Betriebssysteme ist der Speicherplatz bei Chipkarten so stark eingeschränkt, dass vielfach nicht alle genormten Kommandos und Dateistrukturen implementiert werden können. Aus diesem Grund hat man in der relevanten Norm für generell einsetzbare Betriebssysteme ISO/IEC 7816-4 so genannte Profile für Chipkarten eingeführt. Diese definieren jeweils eine Untermenge von Kommandos und Dateistrukturen der entsprechenden Norm.

Die Untermenge muss dann in einer Chipkarte des bestimmten Profils mindestens enthalten sein. Die Beschreibung der Kartenprofile ist aber in beiden Normen im Anhang als informativ und nicht als normativ gekennzeichnet, sodass sie nur eine Empfehlung an die Betriebssystemdesigner darstellen. Die in der ISO/IEC 7816-4 aufgeführten fünf Profile sind in Tabelle 5.3 im Überblick beschrieben.

Die auf dem Markt erhältlichen Chipkarten-Betriebssysteme unterstützen in der Regel unterschiedliche Chipkarten-Mikrocontroller mit differierenden Speichergrößen. Aus diesem Grund gibt es in der Praxis ebenfalls oft Betriebssystemprofile, die dann einen bestimmten Funktionsumfang in Abhängigkeit vom Chiptyp festlegen. Diese Profile innerhalb eines Chipkarten-Betriebssystems sind in der Regel so gestaltet, dass zumindest von den kleineren Speichergrößen zu den größeren hin eine einfache Migration von Anwendungen möglich ist, ohne dass Kommandos oder Dateiaufbau geändert werden müssen.

Tabelle 5.3 Kurzbeschreibung der verschiedenen Profile von Chipkarten nach ISO/IEC 7816-4. Die aufgeführten Dateistrukturen und Kommandos stellen jeweils das geforderte Minimum dar.

Profil	**Beschreibung**	
Profil M	Dateistrukturen:	• transparent • linear fixed
	Kommandos:	• READ BINARY, UPDATE BINARY ohne implizite Selektion und maximale Länge bis 256 Byte • READ RECORD, UPDATE RECORD ohne implizite Selektion • SELECT FILE unter direkter Angabe des FID • VERIFY • INTERNAL AUTHENTICATE
Profil N	wie Profil M mit zusätzlicher Verwendung eines DF Name bei SELECT FILE	
Profil O	Dateistrukturen:	• transparent • linear fixed • linear variable • cyclic
	Kommandos:	• READ BINARY, UPDATE BINARY ohne implizite Selektion und maximale Länge bis 256 Byte • READ RECORD, UPDATE RECORD ohne implizite Selektion • APPEND RECORD • SELECT FILE • VERIFY • INTERNAL AUTHENTICATE • EXTERNAL AUTHENTICATE • GET CHALLENGE
Profil P	Dateistrukturen:	• transparent
	Kommandos:	• READ BINARY, UPDATE BINARY ohne implizite Selektion und maximale Länge bis 64 Byte • SELECT FILE unter direkter Angabe des DF Name • VERIFY • INTERNAL AUTHENTICATE
Profil Q	Datenübertragung:	• Secure Messaging
	Dateistrukturen:	• ---
	Kommandos:	• GET DATA • PUT DATA • SELECT FILE unter direkter Angabe des DF Name • VERIFY • INTERNAL AUTHENTICATE • EXTERNAL AUTHENTICATE • GET CHALLENGE

5.3 Entwurfs- und Implementierungsprinzipien

Bekanntermaßen wirken sich Entwurfsfehler erst während der Implementierung aus und verursachen dort ein Vielfaches der Kosten, die sie bei einem besseren und fehlerärmeren Entwurf verursacht hätten. Doch dies ist bei allen Softwareprojekten eine anerkannte Tatsache. Um solche Fehler zu vermeiden, empfiehlt es sich, einige Prinzipien während des Entwurfs und der Implementierung von Chipkarten-Betriebssystemen zu beachten.

Ein Chipkarten-Betriebssystem ist von seiner Aufgabenstellung her immer ein Sicherheitsbetriebssystem, das Informationen verwalten und vor allem geheim halten muss. Ebenso sind in der Regel keinerlei Änderungen oder Updates der Software während des Betriebs mehr möglich. Damit ist das oberste Prinzip schon vorgegeben. Ein Chipkarten-Betriebssystem muss extrem zuverlässig und damit auch extrem fehlerarm sein. Eine totale Fehlerfreiheit lässt sich in der Realität ja nie erreichen, da selbst die kleinen Betriebssystemkerne der Chipkarten zu groß sind, um sie vollständig in allen Möglichkeiten des internen Programmablaufs zu prüfen.

Ein streng modularer Aufbau trägt aber entscheidend dazu bei, dass eventuelle Fehler während der Implementierungsphase entdeckt und beseitigt werden können. Diese Modularität, die die Zuverlässigkeit stark erhöht, muss nicht unbedingt mit einem großen Mehraufwand an Programmcode verbunden sein. Ein weiterer Vorteil der Modularität ist die Tatsache, dass sich mögliche Systemzusammenbrüche im Allgemeinen nicht so stark auf die Sicherheit auswirken wie bei einem hochoptimierten und speichersparenden Programmcode. Dies wiederum führt dazu, dass die Auswirkungen möglicher Fehler lokal bleiben und das Betriebssystem als Ganzes robuster und stabiler wird.

Dadurch, dass die Implementation in Assembler oder Hardware-nahem C durchgeführt werden muss, erhöht sich auch die Fehleranfälligkeit. Der Aufbau aus einzeln vollständig austestbaren Modulen trägt durch die definierten Schnittstellen stark dazu bei, Programmierfehler rechtzeitig zu erkennen und einzugrenzen. Dies führt in der Konsequenz zu dem in Bild 5.2 dargestellten Schichtenaufbau des Betriebssystems. Der höhere Planungs- und Codierungsaufwand wird durch die erheblich einfacheren Tests und Prüfungen wirtschaftlich auf jeden Fall wettgemacht. Dies hat dazu geführt, dass mittlerweile fast alle Betriebssysteme die hier beschriebene oder dazu eine sehr ähnliche interne Struktur besitzen.

Die übliche Vorgehensweise bei dem Systementwurf ist das Modul-Schnittstellen-Konzept. Während des Entwurfs werden die Aufgaben des Betriebssystems und der Anwendung so weit wie möglich in Funktionen zerlegt und diese dann in Module zusammengefasst. Sind die Schnittstellen der Module exakt beschrieben, so wird mit der Programmierung der einzelnen Module, auch durch mehrere Personen, begonnen. Im Idealfall erstellt man als Erstes eine plattformfreie Implementierung, die noch keine spezifischen Eigenschaften des jeweiligen Mikrocontrollers besitzt. Nachdem diese vollständig geprüft wurde, kann man die notwendigen Adaptionen an den entsprechenden Mikrocontroller vornehmen.

Dadurch, dass Chipkarten-Betriebssysteme vom Programmcode-Umfang immer noch klein sind, ist diese sehr pragmatische Vorgehensweise ohne größere Probleme einsetzbar. Die Vorteile, nämlich der geringe Planungsbedarf und die Verteilungsmöglichkeit der Programmieraufgaben auf mehrere Personen, sowie die einfache Wiederverwendbarkeit von Programmcode, kommen hier voll zum Tragen. Die Nachteile, die durch dieses Verfahren erkauft werden müssen, nämlich der schwierigere Nachweis der Korrektheit des Systems

und die in manchen Änderungsfällen großen Auswirkungen auf viele Module, stehen diesen Vorteilen gegenüber.

Die Software-Entwicklung von Betriebssystemen für Chipkarten bewegt sich von der kompletten Programmierung in Assembler weg. Beinahe alle neuen Projekte werden von Anfang an in der Hardware-nahen Hochsprache C durchgeführt, der eigentliche Kern der Betriebssysteme baut jedoch weiterhin auf maschinenabhängigen Assemblerroutinen auf, während alle übergeordneten Module – wie Dateiverwaltung, Zustandsautomat und Kommandointerpreter – in C programmiert sind. Dies senkt deutlich die Implementierungszeit, die Programme sind leichter zu portieren, wieder zu verwenden, und vor allem ist der Programmcode dank der Verwendung einer Hochsprache wesentlich besser prüfbar. Diese bessere und übersichtlichere Programmstruktur durch eine Hochsprache führt in der Folge zu einer merklich geringeren Fehlerquote.

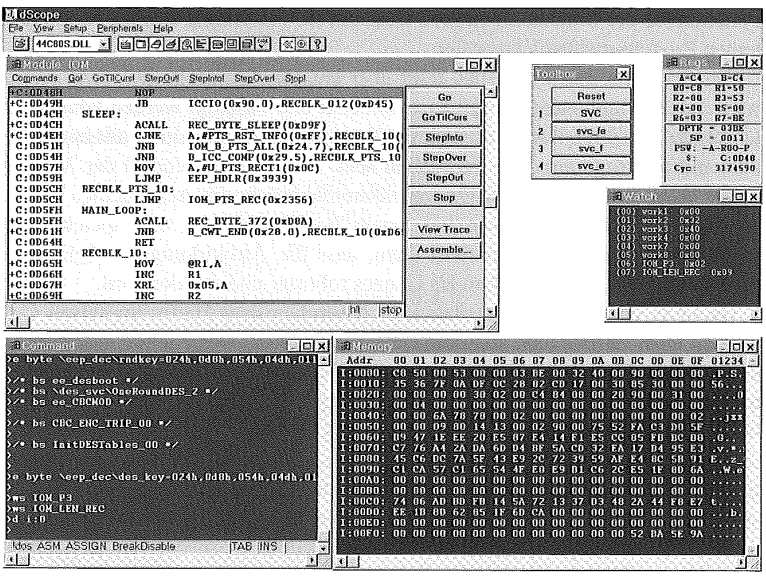


Bild 5.4 Beispiel der Simulation eines Chipkarten-Mikrocontrollers im Umfeld der typischen Entwicklungsumgebung für die Programmiersprachen Assembler und C. Links oben befindet sich das Fenster mit dem Sourcecode; rechts daneben finden Sie Anzeigen für verschiedene Prozessorregister, darunter ein Speicherabbild des RAMs und links unten die Kommandozeile für die Simulatorsteuerung. Der Simulator ermöglicht es dem Softwareentwickler, alle Funktionen des Mikrocontrollers zu überwachen und an jeder Stelle des Programmablaufs einzugreifen. (Keil)

Leider benötigt der Programmcode, den selbst hochoptimierende C-Compiler generieren, für die gleiche Funktionalität zwischen 20 % und 40 % mehr Speicherplatz im ROM als der in Assembler geschriebene Programmcode. Ferner ist die Performanz bei einer Implementierung in C geringfügig niedriger als bei einer Programmierung in Assembler. Heikel ist dies allerdings nur bei kryptografischen Algorithmen und den Übertragungs-

protokollen, da alle anderen Programmteile in Chipkarten-Betriebssystemen in der Regel keine zeitkritischen Abläufe enthalten.

Das größte Problem bei der Programmierung in C ist aber nicht unbedingt der zusätzlich im ROM notwendige Speicherplatz bzw. die niedrigere Ausführungsgeschwindigkeit, sondern der Verbrauch an RAM. Da diese Speicherart aber äußerst begrenzt in einer Chipkarte zur Verfügung steht und sie zu allem Nachteil auch noch im Verhältnis zur Speicherzellengröße am meisten Platz auf dem Chip benötigt, ist dies der Hauptgrund, weswegen Hochsprachen zur Programmierung von Chipkarten-Betriebssystemen bisher eher verhalten eingesetzt werden.

Tabelle 5.4 Einige Beispiele für den typischen Speicherverbrauch von in Assembler implementierten Funktionen eines Chipkarten-Betriebssystems.

Funktion	**benötigter Programmspeicher**
CRC-Algorithmus	≈ 50 Byte
Dateiverwaltung (MF, 2 DF Ebenen, EF, 4 EF Strukturen)	≈ 1200 Byte
DES-Algorithmus (nicht SPA/DPA resistent)	≈ 1200 Byte
DES-Algorithmus (SPA/DPA resistent)	≈ 2000 Byte
EEPROM Schreiben	≈ 150 Byte
RSA-Algorithmus (mit NPU)	≈ 300 Byte
Übertragungsprotokoll T = 0	≈ 500 Byte
Übertragungsprotokoll T = 1	≈ 1200 Byte

Da Chipkarten in sehr sicherheitssensiblen Bereichen Einsatz finden, muss der Kartenherausgeber bzw. Anwendungsanbieter viel Vertrauen in die Integrität des Betriebssystem-Herstellers aufbringen, denn dieser hat jede Möglichkeit, um durch eine vorsätzlich eingebrachte Sicherheitslücke das gesamte System zu diskriminieren. Als Beispiel denke man sich nur eine elektronische Geldbörse auf einer Chipkarte, deren Ladekommando so manipuliert wurde, dass bei einer bestimmten Konstellation die Börse unerlaubt aufgeladen werden kann.

Solche Szenarien sind auch der Grund, warum sich bisher international nur einige Betriebssystem-Hersteller durchsetzen konnten. Das Risiko, sich ein vermeintlich sicheres Betriebssystem mit einem trojanischen Pferd von einem kleinen, unbekannten Hersteller ins Haus zu holen, ist einfach wesentlich größer als bei dem Produkt eines allgemein bekannten Herstellers.

Um jedoch für diese Fälle eine höhere Nachvollziehbarkeit und Sicherheit zu erreichen, werden seit einiger Zeit bei Chipkarten-Betriebssystemen vermehrt Evaluierungen nach ITSEC bzw. dem Nachfolger Common Criteria angestrebt.[2] Dies geschieht entweder freiwillig auf Initiative der Betriebssystem-Hersteller oder wird von den großen Kartenherausgebern gefordert. Diese wollen durch eine Evaluierung des Chipkarten-Betriebssystems die Sicherheit erhöhen, dass keine signifikanten Fehler im Programmcode enthalten sind. Eine Prüfung auf absichtlich eingeschleuste trojanische Pferde würde wahrscheinlich auch durch eine Evaluierung nur bedingt entdeckt werden, da die Möglichkeiten dazu praktisch unbegrenzt sind.

[2] Siehe auch Abschnitt 9.3 „Evaluierung und Test von Software".

Bisher übliche Evaluierungsstufen bei Chipkarten-Betriebssystemen sind E3 und E4 nach ITSEC. Vereinzelt wird sogar die Evaluierungsstufe E6 gefordert und auch angeboten. Dabei darf jedoch nicht übersehen werden, dass eine Evaluierung nach E4 für ein vollständiges mittelgroßes Chipkarten-Betriebssystem im Bereich von 300000 Euro und darüber liegen kann. Hinzu kommt dann noch die Pflicht einer Neuevaluierung bei Änderungen im Programmcode, was freilich weniger aufwendig ist als die Erstevaluierung, da dann in der Regel nur eine so genannte Delta-Evaluierung der Unterschiede durchgeführt werden muss. Dies sind die wesentlichen Gründe, weshalb noch immer verhältnismäßig wenige Chipkarten-Betriebssysteme über eine Evaluierung nach ITSEC oder CC verfügen. Aus diesem Grund werden auch vielfach Evaluierungen von Prüfinstituten ohne einen direkten Bezug zu ITSEC oder CC durchgeführt. Dabei beschränkt man sich auf eine gründliche Prüfung der Designkriterien, des Sourcecodes und der Dokumentation. Dies ist beispielsweise grundsätzlich Pflicht bei den Betriebssystemen für die deutschen ec-Karten.

5.4 Komplettierung

Der Lebenszyklus eines Chipkarten-Betriebssystems verläuft in zwei Teilen – dem Teil vor und dem Teil nach der Komplettierung. Im Abschnitt vor der Komplettierung, bei dem der Mikrocontroller aus der Halbleiterfertigung mit leerem EEPROM kommt, laufen alle Programmteile im ROM ab. Es werden weder Daten aus dem EEPROM gelesen noch dort Programme ausgeführt. Stellt sich zu diesem Zeitpunkt heraus, dass im ROM-Code ein Fehler vorhanden ist, der die Komplettierung unmöglich macht, so muss die gesamte produzierte Charge der Mikrocontroller vernichtet werden, da für die Chips keine weitere Verwendung besteht.

Um die Wahrscheinlichkeit eines solchen Fehlers gering zu halten, könnte man nur eine kleine Laderoutine für das EEPROM in das ROM implementieren und dann das eigentliche Betriebssystem in das EEPROM nachladen. Da aber die erforderliche Chipfläche für ein Bit im EEPROM 4-mal größer ist als für ein Bit im ROM und sich dies somit überproportional auf den Preis des Chips auswirkt, muss sich aus rein wirtschaftlichen Gründen so viel Code wie möglich im ROM befinden. Deshalb sind alle Routinen für den Betriebssystemkern und auch den Rest des Betriebssystems in ihren wesentlichen Teilen komplett im ROM untergebracht. Lediglich einige Aussprünge ins EEPROM sind dann noch für den komplettierten Fall vorgesehen.

Einige Betriebssysteme laufen auch nach der Komplettierung noch vollständig im ROM ab, nur die Daten sind im EEPROM untergebracht, umso die Größe des teuren EEPROMs möglichst klein zu halten. Natürlich erkauft man sich diese Minimierung der verbrauchten Fläche an Speicherplatz durch große Einschränkungen der Flexibilität des Betriebssystems.

Bei der Komplettierung werden dann die ROM-Teile für die eigentliche Anwendung angepasst. Der ROM-Teil ist sozusagen eine große Bibliothek, die durch das EEPROM zu einer funktionsfähigen Anwendung verbunden und ausgebaut wird. Zusätzlich besteht bei fast allen Betriebssystemen noch die Möglichkeit, bei der Komplettierung Programmcode für weitere Kommandos oder spezielle kryptografische Algorithmen in das EEPROM zu laden. Dies ist aber unabhängig von eventuell vorhandenen ausführbaren Dateien, da der Inhalt zu einem späteren Zeitpunkt, z. B. vom Personalisierer, geladen werden kann. Die während der Komplettierung ins EEPROM eingebrachten Programme sind vollständig in das Betriebssystem eingebunden und von diesem auch direkt zu benutzen.

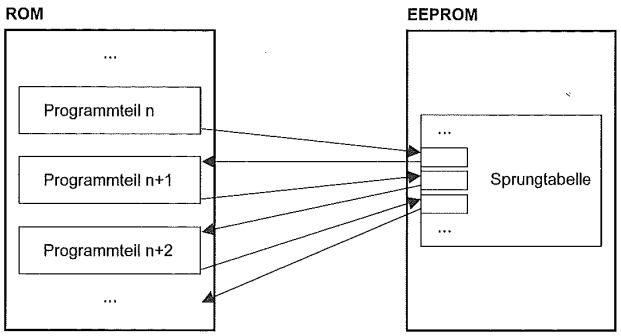


Bild 5.5 Verbindung von Programmcodeteilen im ROM durch eine Verknüpfungstabelle, welche beim Komplettieren des Betriebssystems in das EEPROM geladen wird.

Um ein Chipkarten-Betriebssystem im EEPROM zu komplettieren, ist eine einseitige oder gegenseitige Authentisierung zwischen Chipkarte und der äußeren Welt notwendig. Dazu existieren eine Reihe unterschiedlichster Methoden. Bild 5.6 zeigt dazu als typisches Beispiel einen einfachen, jedoch verhältnismäßig flexiblen Weg. Der Hersteller des Chipkarten-Betriebssystems integriert in das ROM des Mikrocontrollers einen geheimen Wert, der damit charakteristisch für diese ROM-Version des Betriebssystems ist. Bei der Chipfertigung schreibt der Halbleiterhersteller einen geheimen Schlüssel in das EEPROM, welcher beispielsweise für jede Fertigungscharge einen anderen Wert hat. Mit Kenntnis des ROM- und EEPROM-Schlüssels und eines Kryptoalgorithmus kann anschließend für die Chipkartenfertigung ein Authentisierungswert berechnet werden. Das Betriebssystem kann nur nach erfolgreicher Authentisierung komplettiert werden.

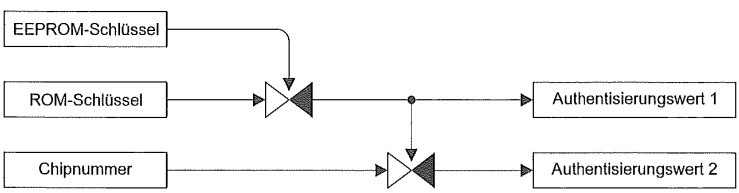


Bild 5.6 Ein mögliches Schema zur Aufteilung und Ableitung der geheimen Schlüssel für die Authentisierung vor der Komplettierung des Chipkarten-Betriebssystems. Der Authentisierungswert 1 ist individuell für eine Charge von Mikrocontrollern, der Authentisierungswert 2 dagegen ist für jeden Mikrocontroller individuell. Als Kryptoalgorithmus könnte beispielsweise der Triple-DES oder AES zum Einsatz kommen.

Mit diesem Schema ist es nun möglich, mit chargenspezifischen Authentisierungswerten zu arbeiten. Würde beispielsweise ein Betriebssystemhersteller einem Kartenhersteller nur einen Authentisierungswert liefern, so könnte dieser nur genau eine spezielle Charge von Chipkarten-Mikrocontrollern komplettieren. Der Vorteil besteht darin, dass der Betriebssystemhersteller dabei nur eine einzige ROM-Maske benötigt und nicht für jeden Kartenhersteller eine ROM-Variante bilden muss.

Dieses Schema lässt sich noch weiter verfeinern. So könnte unter Zuhilfenahme der Seriennummer des Mikrocontrollers ein chipindividueller Authentisierungswert erzeugt werden. Dies würde einerseits die Sicherheit erhöhen, falls einmal Chips auf dem Wege zur Komplettierung abhanden kommen, andererseits kann der Betriebssystemhersteller damit die Komplettierung stückgenau kontrollieren. Der Komplettierer erhält dabei eine Liste der Authentisierungswerte und kann anschließend nur bei den entsprechenden Mikrocontrollern das Betriebssystem vervollständigen. Dies ist im Übrigen ein in dieser oder ähnlicher Form praxisübliches Verfahren.

Es wäre vorstellbar, dass man in einer Chipkartenfertigung eine spezielle Chipkarte mit eingebauter Speicherlesefunktion einschleust, diese komplettiert, durch Initialisierung und Personalisierung schleust und anschließend aus dem Speicher die in den vorhergehenden Schritten eingetragenen geheimen Daten ausliest. Aufgrund der hohen Sicherheitsmaßnahmen wäre das bei den typischen Kartenfertigungen zwar ziemlich schwierig, doch sicherlich nicht völlig auszuschließen. Um nun diesen denkbaren Angriff von vornherein abzuwehren, kann man vor der Komplettierung prüfen, ob ein echter Mikrocontroller vorliegt. Zur Lösung dieses Problems sind eine Vielzahl möglicher Verfahren denkbar. Eines ist im folgenden Bild 5.7 kurz vorgestellt.

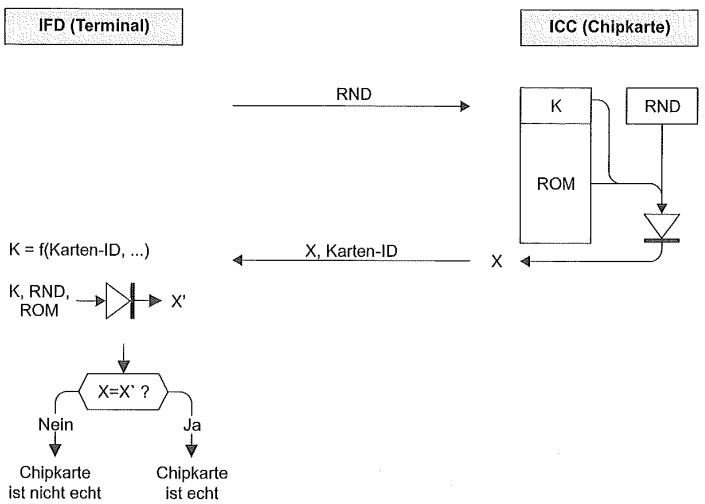


Bild 5.7 Möglicher Ablauf, um sicherzustellen, dass der ROM-Anteil eines Chipkarten-Betriebssystems authentisch ist. Eine diesbezügliche Überprüfung ist vor allem vor der Komplettierung des Betriebssystems im EEPROM sinnvoll, um sicher auszuschließen, dass gefälschte Karten echte Komplettierungsdaten erhalten.

Voraussetzung des Verfahrens ist, dass beim Halbleiterhersteller ein individueller Schlüssel in jeden Mikrocontroller eingetragen wird. Vor der Komplettierung wird nun eine Zufallszahl zur Chipkarte gesendet, welche die Aufgabe hat, einen Replay-Angriff unmöglich zu machen. Diese Zufallszahl, der individuelle Kartenschlüssel und der Inhalt des ROMs werden nun in der Chipkarte mit einer Hash-Funktion in einen Hash-Wert überführt, der mit einer Kartenkennung zur äußeren Welt übertragen wird. Dort kann dann auf-

grund der Kennung der kartenindividuelle Schlüssel errechnet werden. Da der ROM-Inhalt und die Zufallszahl der äußeren Welt bekannt sind, ist es dort möglich, die Hash-Berechnung nachzuvollziehen und mit dem von der Chipkarte übergebenen Hash-Wert zu vergleichen. Sind beide Werte identisch, handelt es sich um eine echte Chipkarte. Dieses Verfahren ist verhältnismäßig einfach, aber durchaus zufriedenstellend wirksam. Es ließe sich in einigen Punkten noch verfeinern, doch soll hier vor allem ein denkbarer Ansatz zur Echtheitsfeststellung vor Komplettierung aufgezeigt werden.

Hardware-Erkennung

Die meisten neueren Betriebssysteme sind auf Mikrocontrollern mit unterschiedlich großen EEPROM-Speichern lauffähig. Die Größe von ROM und RAM muss dabei jedoch gleich bleiben. Somit ist es für einen Kartenherausgeber möglich, immer die für ihn billigste Chipgröße einzusetzen. Er könnte beispielsweise mit einer kostengünstigen Monoapplikationskarte mit 1 kByte EEPROM beginnen und bei Bedarf dann teurere Mikrocontroller mit 4, 16, 32 oder 64 kByte EEPROM für seine Multiapplikationskarten verwenden. Sofern der Halbleiterhersteller diese Bandbreite an Chiptypen unterstützt, muss nur noch das Chipkarten-Betriebssystem eine dazu passende Funktionalität aufweisen. Dies bedeutet, dass es eine automatische Erkennung der Größe des EEPROM-Speichers haben muss und daraufhin seine internen Zeigerstrukturen für den maximalen Freispeicher, Dateigrößen und ähnliche Rahmenparameter setzt. Technisch wird dies realisiert, indem eine Betriebssystemroutine die Fertigungsdaten des Halbleiterherstellers liest und anhand dieser Informationen die Größe des auf diesem Chip zur Verfügung stehenden EEPROM-Speichers berechnet. Das Verfahren funktioniert ausschließlich bei unterschiedlichen Größen des EEPROM-Speichers. Die heutigen Chipkarten-Betriebssysteme sind softwaretechnisch noch nicht in der Lage, sich an variable Speichergrößen von ROM oder RAM anzupassen.

Der große Vorteil solch einer Hardwareerkennung besteht darin, dass der Betriebssystemhersteller den Programmcode nicht mehr auf die veränderten EEPROM-Speichergrößen abzustimmen braucht. Damit wird eine mögliche Fehlerquelle ausgeschlossen, und vor allem ist keine abermalige Evaluierung des Betriebssystems für die neue Hardwareplattform notwendig. Die Hardware-Erkennung der modernen Betriebssysteme spart erheblich Zeit bei der Software-Entwicklung und kann damit die Durchlaufzeit um mehrere Wochen senken.

Soft- und Hardmaske

Im Zusammenhang mit Feldversuchen und Chipkarten-Betriebssystemen werden oft die Begriffe Softmaske (*softmask*) und Hardmaske (*hardmask*) benutzt. Genau genommen und streng logisch gesehen, sind beide Begriffe unsinnig, da eine ROM-Maske – gemeint ist dabei der Programmcode, der sich im ROM befindet – immer unveränderlich, also fest ist. Im Sprachgebrauch der Chipkartenwelt ist aber mit Softmaske nur so etwas Ähnliches wie eine ROM-Maske gemeint. Man spricht von einer Softmaske, wenn sich Teile oder der gesamte Programmcode für ein Chipkarten-Betriebssystem oder die Kommandos einer Anwendung im EEPROM befinden. Der Code lässt sich aufgrund dieser Sache einfach ändern, ohne dass zeit- und kostenintensiv eine neue ROM-Maske erstellt werden muss. Diese Art von „Maske" ist also „weich" und veränderbar und insofern eine Softmaske. Vor allem bei Tests und Feldversuchen wird diese Technik benutzt, da sich kurzfristig und mit geringem Aufwand Fehlerbeseitigungen und Programmadaptionen durchführen lassen.

Von Nachteil dabei ist jedoch die Tatsache, dass dazu Chips verwendet werden müssen, die ein großes EEPROM besitzen und deshalb teurer sind als äquivalente Chips mit dem Programmcode im ROM. Da bei Feldversuchen jedoch üblicherweise keine Millionenstückzahlen an Karten ausgegeben werden, sind die erhöhten Kosten für die Chips mit größerem EEPROM-Speicher durchaus vertretbar.

Ist dann jedoch der Test oder Feldversuch mit der Softmaske abgeschlossen und der im EEPROM befindliche Programmcode ohne weitere Nachbesserungen lauffähig, kann man den Programmcode vom EEPROM ins ROM verlagern und eine echte ROM-Maske herstellen. Dies ist mit nur geringem Arbeitsaufwand durchführbar und wird dann als „Hardmaske" bezeichnet, da sie „hart" und unveränderlich ist. Der Vorteil der Hardmaske besteht genau genommen nur darin, dass die gleiche Speichergröße im ROM wesentlich weniger Chipfläche als im EEPROM beansprucht und damit für die gleiche Menge an Programmcode kleinere und billigere Mikrocontroller verwendet werden können.

Das zweistufige Vorgehen mit Softmaske und Hardmaske bei neuen Chipkarten-Anwendungen schafft zusätzliche Flexibilität und die Chance, auch noch kurz vor der Ausgabe der Chipkarten an die Benutzer substanzielle Änderungen am Programmcode vornehmen zu können. Bei der klassischen Vorgehensweise mit einer reinen ROM-Maske sind am Programmcode in der Chipkarte nach Maskenabgabe keine größeren Änderungen mehr durchführbar. Die Überlegenheit dieser Vorgehensweise gegenüber dem althergebrachten Ablauf hat dazu geführt, dass mittlerweile beinahe jede Einführung einer neuen Chipkarten-Anwendung zunächst mit einer Softmaske startet und anschließend nach unter Umständen notwendigen Adaptionen des Programmcodes auf eine Hardmaske migriert wird.

APIs von Betriebssystemen

Ursprünglich boten Chipkarten-Betriebssysteme keine Möglichkeit, selbst erstellten Programmcode zu laden und bei Bedarf auf der Chipkarte auszuführen. Deshalb besaßen die herkömmlichen Betriebssysteme keine veröffentlichte Programmierschnittstelle, die für Aufrufe von Betriebssystemfunktionen von Dritten genutzt werden konnte. Durch die neueren Entwicklungen der Chipkarten-Betriebssysteme wie beispielsweise Java unterstützende Betriebssysteme (Java Card) oder MULTOS wurde die Möglichkeit geschaffen, eigenen Programmcode auf die Chipkarte zu laden. Um nicht bereits im Betriebssystem vorhandene Programmroutinen nochmals nachprogrammieren zu müssen, besitzen diese Betriebssysteme ein durchdachtes API (*application programming interface*), das Zugriffe auf die wichtigsten Funktionen des Betriebssystems gestattet. Zwar haben praktisch alle Chipkarten-Betriebssysteme eigene, interne APIs, doch sind diese nicht für die Verwendung durch Externe konzipiert und in der Regel vertraulich.

Ausnahmsweise existiert, wie sonst in der Chipkartenwelt üblich, bei APIs für Chipkarten-Betriebssysteme keine Norm, sondern es haben sich bislang vor allem zwei Industriestandards etablieren können: Dies sind zum einen die verschiedenen Java Card APIs und zum anderen das API von Multos. Die in den dazugehörigen Spezifikationen beschriebenen APIs lassen Zugriffe auf die wesentlichen Funktionen des Betriebssystems zu.[3] Dazu gehören Zugriffe auf die Dateiverwaltung, Aufrufe der vorhandenen kryptografischen Funktionen und natürlich auch das Senden und Empfangen von Daten. Ein Chipkar-

[3] Siehe auch Abschnitt 5.14.1 „Java Card" und 5.14.2 „Multos".

ten-Betriebssystem, das ein vollständiges Betriebssystem-API hat und auch Programmcode nachladen kann, unterscheidet sich im Wesentlichen nur in dem zur Verfügung stehenden Speicher von den bekannten PC-Betriebssystemen.

5.5 Speicherorganisation

Die drei verschiedenen Speicherarten eines Chipkarten-Mikrocontrollers haben völlig unterschiedliche Eigenschaften. Das ROM kann nur während der Herstellung des Mikrocontrollers als Maske im Gesamten programmiert werden und ist dann während der ganzen Lebensdauer der Chipkarte statisch. Aufgrund des Aufbaus ist die Wahrscheinlichkeit der ungewollten Veränderung von ROM-Inhalten praktisch null.

Im Gegensatz dazu bleibt der Inhalt des RAM nur so lange erhalten, wie Spannung an der Chipkarte anliegt. Ein Spannungsausfall verursacht den totalen Verlust aller im RAM enthaltenen Daten. Allerdings können die Daten im RAM mit voller Arbeitsgeschwindigkeit des Prozessors geschrieben und auch unbegrenzt oft wieder gelöscht werden.

Das EEPROM kann hingegen auch ohne äußere Spannungszufuhr Daten speichern. Es hat allerdings die drei Nachteile, dass es eine begrenzte Lebensdauer hat, in Seiten (*pages*) aufgeteilt ist und Schreib- und Löschzugriffe sehr lange dauern (ca. 1 ms/Byte).

Der im ROM für das Betriebssystem enthaltene Programmcode muss bis auf die fest vom Mikrocontroller vorgegebenen Interruptvektoren keiner bestimmten Aufteilung gehorchen. Die einzelnen Programmteile können also in beliebiger Reihenfolge miteinander verbunden werden, wobei immer das eine Ziel besteht, Sprünge in der Software in ihrer Entfernung zu minimieren, um auf diese Weise Speicherplatz einzusparen. Wichtig ist, dass das ROM durch Fehlererkennungscodes (EDCs) abgesichert ist, da Fehler im ROM durchaus einmal auftreten können. So führt beispielsweise ein Kratzer in der ROM-Sektion des Mikrocontrollers oder ein Abbruch eines Teils des Chips beim Bonden zu fehlerhaften Daten im ROM. Interessanterweise bedeutet dies nicht zwangsläufig, dass das Betriebssystem nicht mehr funktionsfähig ist, sondern es kann durchaus sein, dass nur einige Programmteile fehlerhaft ablaufen. Um so einen Fall schon im Ansatz zu verhindern, wird beim Starten des Chipkarten-Betriebssystems das ROM grundsätzlich auf Fehlerfreiheit geprüft.

Die folgende Übersicht in Bild 5.8 zeigt eine übliche Aufteilung von 256 Bytes großen RAMs. Es ist aufgeteilt in die Bereiche Register, Stack, allgemeine Variablen, Arbeitsbereiche für kryptografische Algorithmen und I/O-Puffer. Ist beispielsweise ein I/O-Puffer mit der Länge 256 Byte gefordert oder müssen zusätzliche Variablen im RAM untergebracht werden, sind sehr schnell die zur Verfügung stehenden Speichergrenzen erreicht. Das Problem wird durch einen Arbeitsbereich im EEPROM gelöst, der wie ein RAM verwendet wird. Nachteilig ist, dass dabei die Schreibzeiten rund um den Faktor 10000 größer sind als beim einem RAM-Zugriff. Ein zweiter Nachteil ist die begrenzte Lebensdauer einer EEPROM-Zelle, die im Gegensatz zu RAM-Zellen nicht unbegrenzt oft beschrieben werden kann. Allerdings ist die Auslagerung von RAM-Inhalten ins EEPROM oft der einzige Lösungsweg, wenn beispielsweise I/O-Puffer gefordert werden, die größer sind als das insgesamt zur Verfügung stehende RAM. Zur Gegenüberstellung enthält das Bild auch noch die typische Aufteilung eines 6 kByte großen RAM für eine sehr leistungsfähige Chipkarte für Telekommunikationsanwendungen.

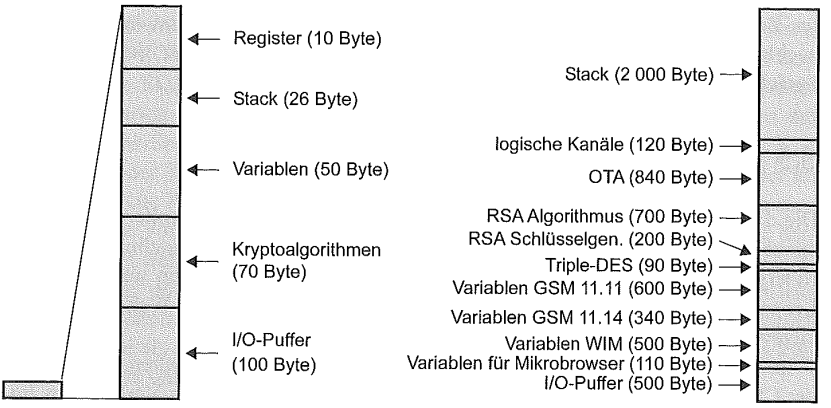


Bild 5.8 Die linke Seite zeigt eine typische Aufteilung des 256 Byte großen RAMs eines einfachen Chipkarten-Betriebssystems, programmiert in Assembler. Rechts befindet sich das Speicherlayout eines 6 kByte großen RAMs für eine leistungsfähige Telekommunikations-Chipkarte, deren Betriebssystem in der Programmiersprache C erstellt wurde.

Die Datenablage im EEPROM ist ungleich komplizierter und aufwendiger als die der beiden anderen Speicherarten. Die Basisaufteilung ist bei einem modernen Betriebssystem in etwa folgendermaßen: In einem bei vielen Mikrocontrollern durch die Hardware besonders geschützten Bereich am Anfang des EEPROM können diverse Fertigungsdaten, z. B. eine nur einmal verwendete und somit einzigartige Nummer, in den Chip geschrieben werden. Viele Halbleiterhersteller kennzeichnen in diesem Bereich auch den Chiptyp und die Größe des dem Betriebssystem zur Verfügung stehenden EEPROMs. Meist sind diese Bereiche auch nur für einen WORM (*write once, read multiple*) Zugriff ausgelegt, d. h. man kann sie nur ein einziges Mal beschreiben und anschließend nur noch lesen. Technisch wird dies in der Regel durch übliche EEPROM-Zellen erzielt, die jedoch elektrisch nicht mehr gelöscht werden können.

Auf diesen Bereich, der in der Regel eine Größe von 16 oder 32 Byte hat, folgen die Tabellen und Zeiger für das Betriebssystem. Diese werden bei der Komplettierung in das EEPROM geladen und ergeben zusammen mit den ROM-Programmen erst das vollständige Betriebssystem für die Chipkarte. Um das Betriebssystem immer in einem sicheren und stabilen Zustand betreiben zu können, ist dieser Bereich durch eine Prüfsumme (*error detection code – EDC*), die vor dem Ersten oder sogar jedem Zugriff nachgerechnet wird, abgesichert. Die Entdeckung eines Speicherfehlers bei der Überprüfung muss dazu führen, dass die betroffenen EEPROM-Teile im Folgenden nicht mehr benutzt werden, da eine korrekte Funktion des Betriebssystems nicht mehr sichergestellt ist.

Auf den abgesicherten Teil des Betriebssystems folgt ein Bereich, der zusätzlichen Programmcode für die Anwendungen enthält. Unter Umständen ist auch dieser Bereich mit einer Prüfsumme gegen Veränderungen gesichert. In dem Bereich können anwendungsspezifische Kommandos oder Algorithmen geladen werden, die nicht im ROM stehen sollen oder aus Speicherplatzmangel dort keinen Platz mehr gefunden haben.

Tabelle 5.5 Beispiel für Herstellerdaten, wie sie von manchen Halbleiterherstellern bei der Fertigung in einen WORM-Bereich des EEPROMs geschrieben werden. Die ersten fünf Herstellerdaten zusammengefasst ergeben hier eine 8 Byte lange einzigartige Chipnummer. Der große Vorteil einer mit dieser Methode generierten Chipnummer besteht darin, dass dazu beispielsweise keine hochgenaue und auf verschiedene Produktionsstandorte abgestimmte Uhrzeit notwendig ist, sondern lediglich Daten, die in jeder Halbleiterfertigung an allen Produktionsmaschinen zur Verfügung stehen.

Datenelement	Größe
Halbleiterhersteller	1 Byte
Produktionsstätte	1 Byte
Batch-Nummer in der Halbleiterfertigung	2 Byte
x-Koordinaten auf dem Wafer	2 Byte
y-Koordinaten auf dem Wafer	2 Byte
Mikrocontrollertyp	2 Byte
optionale Hardware-Komponenten	6 Byte
Speichergröße des RAM	2 Byte
Speichergröße des EEPROM	3 Byte
Speichergröße des ROM	3 Byte

Der sich daran anschließende Dateibereich beherbergt alle Dateistrukturen, also den gesamten nach außen hin sichtbaren Dateibaum. Dieser Bereich ist nicht als Ganzes mit einer Prüfsumme geschützt, sondern meist stark dateiorientiert gesichert. Die interne Struktur ist in Bild 5.9 nochmals genauer erläutert.

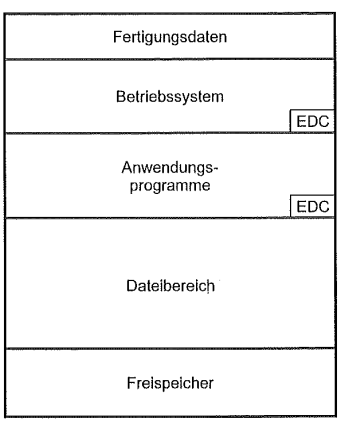


Bild 5.9 Beispiel für die Aufteilung des EEPROM in einem Chipkarten-Betriebssystem.

Am Ende des EEPROMs folgt ein eventuell vorhandener Freispeicher, dessen Verwaltung ein eigener Manager durchführt. Oft ist jedoch dieser Freispeicher einzelnen Anwendungen im Dateibereich zugeordnet und kann dort innerhalb der Anwendungen für neu zu erzeugende Dateien verwendet werden. Ansonsten gehört er zum allgemeinen Dateibereich, wo er nur für neue Anwendungen, die als Ganzes geladen werden, zur Verfügung steht.

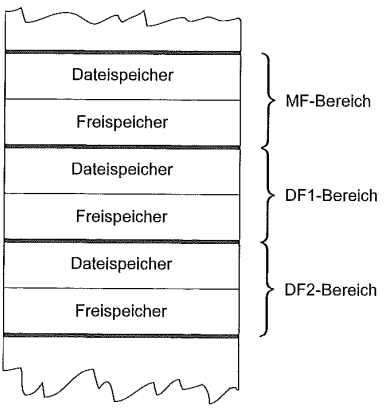


Bild 5.10 Beispiel für die Strukturierung des Dateibereichs Chipkarten-Betriebssystem, das mehrere unabhängige und physikalisch getrennte Anwendungen unterstützt. Dieses Konzept setzt zwar eine extrem rigide Trennung der einzelnen Anwendungen um, es wird jedoch mittlerweile wegen der zu unflexiblen Freispeicherverwaltung nicht mehr in Betriebssystemen mit dynamischer Dateiverwaltung eingesetzt.

5.6 Dateien in der Chipkarte

Neben den in ihnen enthaltenen Mechanismen zur Identifizierung und Authentisierung sind Chipkarten vor allem auch Datenspeicher. Dabei haben sie gegenüber anderen Speichermedien, wie beispielsweise Disketten, den entscheidenden Vorteil, dass der Zugriff auf die Daten an Bedingungen geknüpft sein kann.

Die ersten Chipkarten wiesen nur mehr oder minder direkt adressierbare Speicherbereiche auf, in die Daten geschrieben oder aus denen Daten gelesen werden konnten. Der Zugriff erfolgte unter Angabe von physikalischen Speicheradressen. Mittlerweile verfügen alle Chipkarten über ein vollständiges und hierarchisch organisiertes Dateiverwaltungssystem mit symbolischer und Hardware-unabhängiger Adressierung.

Allerdings weisen diese Dateiverwaltungen einige chipkartenspezifische Eigenheiten auf: Das Auffälligste dabei ist, dass keine Mensch-Maschine-Schnittstelle vorhanden ist. Alle Dateien werden mit hexadezimalen Codes adressiert, und auch die weiteren Kommandos bauen strikt darauf auf, dass hier die Kommunikation nur zwischen zwei Computern abläuft. Ebenfalls typisch für diese Dateiverwaltungen ist, dass sie auf wenig Speicherverbrauch hin ausgelegt sind. Wenn möglich, wird auch noch das letzte redundante Byte vermieden. Da der „Benutzer" im Terminal ein Computer ist, entstehen dadurch eigentlich keine Nachteile.

Um den Speicherverbrauch so gering wie möglich zu halten, ist üblicherweise auch keine aufwendige Speicherverwaltung vorhanden. Wird eine Datei gelöscht – und selbst dies können erst wenige Betriebssysteme –, dann ist es ebendarum nicht selbstverständlich, dass der frei werdende Platz von einer neu zu erzeugenden Datei eingenommen werden kann. Üblicherweise werden bei der Initialisierung bzw. Personalisierung alle Dateien erzeugt und in die Chipkarte geladen. Danach sind Änderungen auf den Dateiinhalt beschränkt.

Die Eigenheiten des benutzten Speichers beeinflussen natürlich auch die Art und Weise der Dateiverwaltung. Die Speicherseiten im EEPROM können im Gegensatz zu PC-Festplatten nicht unbegrenzt geschrieben bzw. gelöscht werden. Dies führt dazu, dass es spezielle Dateiattribute gibt, um Informationen redundant und gegebenenfalls sogar korrigierbar abzulegen.

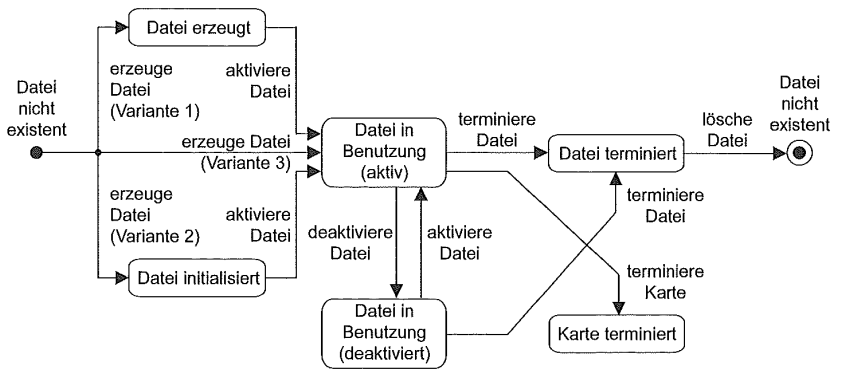


Bild 5.11 Zustände und Zustandsübergänge im vollständigen Lebenszyklus von Dateien in einem Chipkarten-Betriebssystem nach ISO/IEC 7816-9. Die unterschiedlichen Zustände nach der Erzeugung einer Datei resultieren aus möglichen Parametern.

Interner Aufbau von Dateien

Die neueren Dateiverwaltungssysteme für Chipkarten sind objektorientiert aufgebaut. Dies bedeutet, dass alle Informationen über eine Datei in dieser Datei selbst gespeichert sind. Eine weitere Auswirkung dieses Prinzips besteht darin, dass eine Datei vor einer Aktion immer zuerst selektiert werden muss. Dateien sind deswegen in diesen Systemen immer in zwei getrennte Teile aufgespalten. Der Dateiheader genannte Teil enthält Informationen über die Struktur, den Aufbau der Datei und Zugriffsbedingungen, während in dem mit einem Zeiger verbundenen Dateibody die veränderbaren Nutzdaten gespeichert sind.

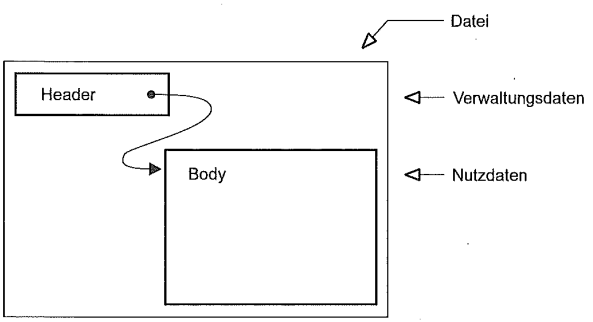


Bild 5.12 Der interne Aufbau einer Datei bei Dateiverwaltungen von Chipkarten.

Dieser Aufbau hat neben der besseren Strukturierung der Daten überdies den Vorteil der größeren physikalischen Sicherheit der Datenbestände. Das in Speicherseiten (*pages*) orientierte EEPROM, in dem sich alle Dateien befinden, weist nur eine begrenzte Anzahl von Schreib-/Löschzyklen auf. Header und Body befinden sich stets auf getrennten Seiten des Speichers. Da die Header im Regelfall aber selten verändert werden, dort aber alle Zugriffsbedingungen gespeichert sind, kann durch einen Schreib- oder Löschfehler im Body keine Beeinflussung auftreten. Befänden sich Header und Body auf einer gemeinsamen Speicherseite, wäre es beispielsweise möglich, durch gezielt erzeugte Schreibfehler die Zugriffsbedingungen so zu verändern, dass eventuell geheime Daten aus dem Body ausgelesen werden könnten.

Manche Chipkarten-Betriebssysteme bieten die Möglichkeit, von zwei verschiedenen Headern aus einen Body zu adressieren. Die beiden Header befinden sich dann üblicherweise in DFs zweier unterschiedlicher Anwendungen. Auf diese Weise können technisch elegant Daten zwischen zwei Anwendungen geteilt werden. Wichtig dabei ist, dass die Zugriffsbedingungen, festgelegt in den beiden Headern, identisch sind.

5.6.1 Dateitypen

Der Aufbau von Chipkarten-Dateisystemen ist in der ISO/IEC 7816-4 festgelegt und ähnlich dem von DOS oder Unix. Der größte Unterschied besteht darin, dass bei Chipkarten keine anwendungsspezifischen Dateien existieren, wie zum Beispiel spezielle Dateien für eine bestimmte Textverarbeitung. Bei Chipkarten können nur die genormten Dateistrukturen benutzt werden.

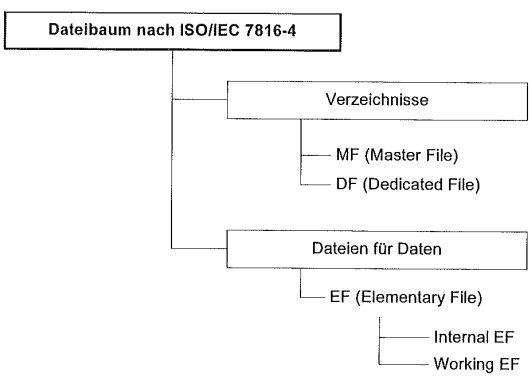


Bild 5.13 Klassifizierungsbaum der Dateitypen bei Chipkarten-Betriebssystemen nach ISO/IEC 7816-4.

Es existieren bei Chipkarten grundsätzlich zwei Kategorien von Dateien. Die Verzeichnisdateien, „Dedicated Files" (DFs) genannt, und die „Elementary Files" (EFs), welche die eigentlichen Nutzdaten beinhalten. Die DFs fungieren als eine Art Ordner und beinhalten entweder weitere, untergeordnete DFs oder EFs, die logisch zusammengehören. Die EFs lassen sich noch in Dateien für die äußere Welt (*working EF*) und solche für das Betriebssystem (*internal EF*) typisieren.

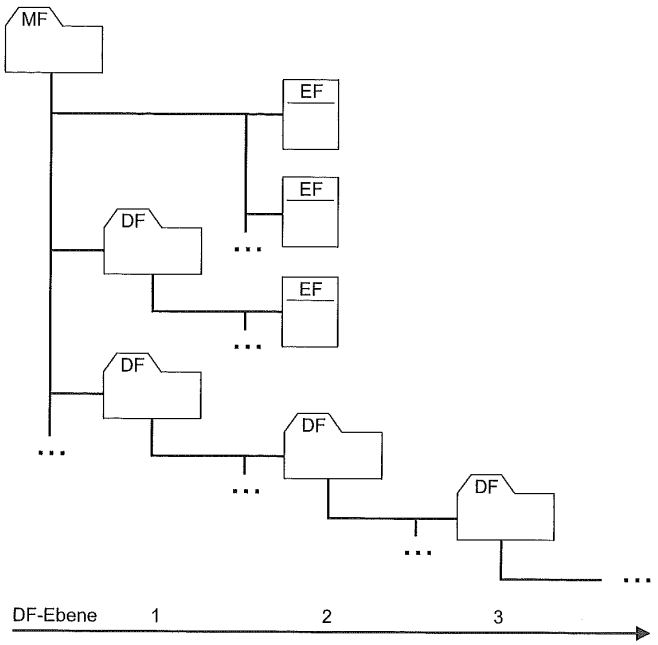


Bild 5.14 Die verschiedenen Dateitypen im Dateibaum einer Chipkarte.

MF

Das Root-Verzeichnis, das implizit nach einem Reset der Chipkarte selektiert wird, hat die Bezeichnung „Master File", abgekürzt MF. In ihm befinden sich alle anderen Verzeichnisse und alle Dateien. Das Master File ist ein Sonderfall eines Dedicated File und stellt den gesamten in der Chipkarte für den Dateibereich verfügbaren Speicher dar. Es muss in jeder Chipkarte vorhanden sein.

DF

Unter dem MF können bei Bedarf noch „Dedicated Files", abgekürzt DFs, existieren. Diese werden vielfach auch „Directory Files" genannt, obwohl dies im Widerspruch zur offiziellen ISO/IEC 7816-4-Abkürzung steht. DFs sind Verzeichnisse, in denen weitere Dateien (EFs und DFs) zusammengefasst sein können. Unter den DFs können auch noch weitere DFs existieren. Im Prinzip ist die Schachtelungstiefe der DFs nicht limitiert. Der in Chipkarten beschränkte Speicherplatz führt jedoch dazu, dass selten mehr als zwei Ebenen von DFs unter dem MF aufgebaut werden.

Bei der Spezifikation für die UICC (TS 102.221)wurde noch eine besondere Form eines DFs eingeführt, das ADF (*application dedicated file*). Dabei handelt es sich um ein DF für Anwendungen, das mit den entsprechenden Mechanismen (SELECT Kommando mit AID) selektiert werden kann, jedoch nicht unter dem MF angeordnet ist. Ein ADF könnte damit auch als eine Art MF beschrieben werden.

EF
Die Nutzdaten, die für eine Anwendung notwendig sind, befinden sich in den EFs. EF ist die Abkürzung von Elementary File. Sie können direkt unter dem MF oder auch unter einem DF angeordnet sein. Um Daten sowohl logisch optimal strukturiert als auch speicherplatzminimiert ablegen zu können, besitzen EFs grundsätzlich eine interne Dateistruktur. Dies ist der Hauptunterschied zu Dateien auf PCs, deren interne Struktur durch eine Anwendung, beispielsweise eine Textverarbeitung, vorgegeben ist und nicht direkt durch das Betriebssystem. EFs sind in Working EFs und Internal EFs aufgeteilt.

Working EF
Alle Daten einer Anwendung, welche vom Terminal gelesen oder geschrieben werden müssen, also für die äußere Welt (vom Standpunkt der Chipkarte aus betrachtet) bestimmt sind, befinden sich in den so genannten Working EFs. Daten, die sich in einem solchen Dateityp befinden, werden nicht vom Betriebssystem benutzt.

Internal EF
Zusätzlich zu den EFs für Anwendungen existieren noch interne Systemdateien, in denen Daten für das Betriebssystem selber, den Ablauf einer Anwendung, geheime Schlüssel oder Programmcode abgelegt werden. Der Zugriff auf diese Dateien ist vom Betriebssystem besonders geschützt. Hier gibt es nun aber nach ISO/IEC 7816-4 zwei Varianten, um diese internen Systemdateien in der Dateiverwaltung zu integrieren. Die Erste sieht vor, diese Dateien im jeweiligen Anwendungs-DF versteckt anzuordnen. Sie können somit auch nicht selektiert werden und die Verwaltung dafür übernimmt völlig transparent das Betriebssystem der Chipkarte ähnlich den Ressource-Dateien im MacOS. In der zweiten Variante erhalten diese Systemdateien einen regulären Filenamen (d. h. eine FID) und sind mit diesem auch selektierbar. Das entspricht in seinen wesentlichen Ansätzen der Dateiverwaltung unter DOS. Beide Systeme haben gleichermaßen einige Vor- und Nachteile, doch erfüllen sie auf etwas unterschiedliche Weise die gleiche Funktionalität.

Dateien für eine Anwendung
Nach einer Konvention fasst man alle Dateien mit Nutzdaten, d. h. EFs, die zu einer Anwendung gehören, immer in einem DF zusammen. Damit erhält man eine klare und übersichtliche Struktur, und durch die Erzeugung des betreffenden DFs kann auf einfache Art eine neue Anwendung in eine Chipkarte eingebracht werden.

Da das MF nur der Sonderfall eines DFs ist, können selbstverständlich bei Chipkarten mit nur einer Anwendung alle dazugehörigen EFs direkt unter dem MF kumuliert werden. In einer typischen Chipkarte für eine einzige Anwendung können sich deshalb alle EFs entweder direkt unter dem MF oder unter dem einzigen DF befinden. Chipkarten mit mehreren Anwendungen besitzen entsprechend viele DFs, in denen die zugehörigen EFs eingerichtet sind.

Unter einem solchen Anwendungs-DF können noch weitere DFs angeordnet sein. Zum Beispiel kann ein DF direkt unter dem MF der Anwendung „Verkehrsleitsystem“ gewidmet sein. Eine zusätzliche Schachtelung könnte in eigenen DFs im Anwendungs-DF die Dateien für die möglichen Sprachen, z. B. „Englisch“ oder „Deutsch“, enthalten.

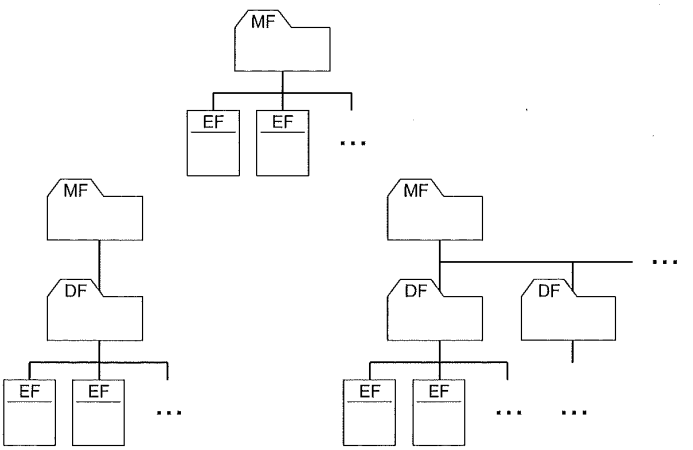


Bild 5.15 Unterschiede in der Dateiorganisation zwischen Chipkarten mit einer und mehreren Anwendungen. Oben und unten links sind die beiden sinnvollen Varianten einer einzigen Anwendung pro Chipkarte aufgezeigt und unten rechts eine Chipkarte mit mehreren Anwendungen separiert in unterschiedlichen DFs.

5.6.2 Dateinamen

Die Dateien in modernen Chipkarten-Betriebssystemen werden ausnahmslos logisch adressiert und nicht über direkte physikalische Adressen angesprochen. Letzteres war früher in der Chipkartenwelt durchaus üblich und ist selbst bei Mikrocontroller-Chipkarten vereinzelt auch heute noch der Fall. Bei einfachen und im Adressbereich genau spezifizierten Anwendungen kann dies eine sehr speicherplatzsparende Zugriffsart sein. Da der Zugriff auf

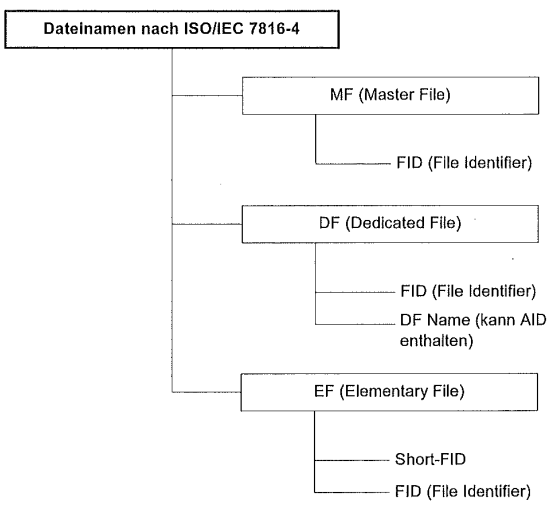


Bild 5.16 Klassifizierungsbaum der Dateinamen bei Chipkarten-Betriebssystemen nach ISO/IEC 7816-4.

alle Dateien von einem Rechner im Terminal ausgeführt wird, bedeutet es auch keinen Verlust an Benutzerfreundlichkeit. Allerdings entspricht eine direkte Adressierung in keiner Weise den Kriterien modernen Software-Designs und verursacht auch große Probleme bei Software-Erweiterungen und Chipkarten-Mikrocontrollern mit unterschiedlichen Adressräumen.

Wesentlich besser und vor allem viel einfacher erweiterbar sind hierbei alle Konzepte mit logischen Adressen der Dateien. Es kann ohne Frage davon ausgegangen werden, dass es in wenigen Jahren keine andere Art der Adressierung von Dateien als die logische über Dateinamen bei Chipkarten mit Mikrocontrollern mehr gibt. Bei Speicherkarten wird es jedoch auf absehbare Zeit bei der physikalischen Adressierung der Daten bleiben.

File Identifier (FID)

Das hier beschriebene System fundiert auf der ISO/IEC 7816-4 und wird im Prinzip auch bei allen anderen internationalen Chipkarten-Normen abgebildet. Alle Dateien, einschließlich der Verzeichnisse, besitzen einen 2 Byte langen File Identifier (FID), unter dessen Verwendung sie ausgewählt werden können.

Das MF hat aus historischen Gründen den FID '3F00', der im ganzen logischen Adressraum ausschließlich dem MF vorbehalten ist. Der logische Dateiname 'FFFF' ist für zukünftige Anwendungen reserviert und darf nicht verwendet werden. Daneben gibt es sowohl von ISO als auch einigen anderen Normen weitere reservierte FIDs, die in der Tabelle 5.6 aufgeführt sind.

Die GSM-Anwendung ist ein typisches Beispiel dafür, dass verschiedentlich Teile des FIDs nicht frei wählbar sind. In der GSM 11.11-Spezifikation ist das höherwertige Byte durch den Ort, den die Datei im Dateibaum einnimmt, festgelegt. Die Codierung selber ist historisch gewachsen und stammt noch von den ersten französischen Chipkarten. DFs bei GSM besitzen als Erstes, d. h. höherwertigstes Byte den Wert '7F'. Die EFs direkt unter dem MF haben als erstes Byte des FID den Wert '2F' und EFs unter einem DF den Wert '6F'. Das niederwertige Byte ist durchnummeriert. Diese Festlegung gilt nur für die GSM-Anwendung und besitzt keinen allgemeinen normativen Charakter. Ansonsten kann der Adressbereich des FID von 2 Bytes voll ausgenutzt werden und unterliegt keinen Beschränkungen.

Tabelle 5.6 Die von den wichtigsten Chipkarten-Normen reservierten File Identifiers.

FID	Name und Zweck	Norm
'2F00'	Diese FID ist reserviert für die Datei EF_{DIR} (*directory*) und wird zur Speicherung von Application Identifiers (AIDs) mit dazugehöriger Pfadangabe zur korrespondierenden Anwendung benutzt.	ISO/IEC 7816-4
'2F01'	Diese FID ist reserviert für die Datei EF_{ATR} mit den Erweiterungen zum ATR.	ISO/IEC 7816-4
'3F00'	Das MF ist das Wurzelverzeichnis für alle Dateien einer Chipkarte.	ISO/IEC 7816-4, GSM 11.11, TS 102.221, EMV
'3FFF'	Diese FID ist reserviert für die Dateiselektion durch Pfadangabe.	ISO/IEC 7816-4
'FFFF'	Diese FID ist reserviert für zukünftige Benutzung durch ISO/IEC.	ISO/IEC 7816-4

Das EF_{DIR} hat die FID '2F00', besitzt die Struktur linear fixed und besteht aus mindestens einem Record. Jeder Record ist wiederum ein constructed Datenobjekt, das Informationen über eine bestimmte Anwendung auf der Chipkarte enthält. Typischerweise sind dies die AID und eine textuelle Bezeichnung der jeweiligen Anwendung. Das EF_{DIR} könnte auch noch weitere Daten wie beispielsweise den Pfad zur Anwendung enthalten. Der Zweck des EF_{DIR} ist es, einem Terminal die auf der Chipkarte befindlichen Anwendungen in einem standardisierten Format anzuzeigen.

Tabelle 5.7 Beispiel für einen typischen Aufbau eines EF_{DIR} nach ISO/IEC 7816-4 und ISO/IEC 7816-5, wie er für die UICC festgelegt ist

EF_{DIR}	**Verzeichnis-EF (*dir – directory*)**	
Beschreibung	Diese Datei enthalt Informationen über die auf einer Chipkarte befindlichen Anwendungen.	
Datei	FID = '2F00'; Struktur: linear fixed, Dateigröße: n Bytes Zugriffe: READ: immer; UPDATE: je nach Anwendung, im Allgemeinen jedoch nur Administrator	
Codierung eines Records	Byte 1:	Template für Anwendung '61'
	Byte 2:	Länge des Template für Anwendung (3 . . . 127)
	Byte 3:	Kennzeichen des AID '4F'
	Byte 4:	Länge des AID (1 . . . 16)
	Byte 5 . . . n:	AID
	Byte n + 1:	Kennzeichen der Anwendungsbezeichnung (*application label*) '50'
	Byte n + 2:	Länge der Anwendungsbezeichnung
	Byte n + 3 . . . m:	Anwendungsbezeichnung in ASCII (0 . . . 16)
Beispiel	'61 0F 4F 05 D2 76 00 00 60 50 05 52 61 6E 6B 6C'	
	'61'	⇒ Template für Anwendung
	'0F'	⇒ Länge des Templates ⇒ Länge = 15 Byte
	'4F'	⇒ Kennzeichen des AID
	'05'	⇒ Länge des AID ⇒ Länge = 5 Byte
	'D2 76 00 00 60'	⇒ AID
	'4F'	⇒ Kennzeichen der Anwendungsbezeichnung
	'05'	⇒ Länge der Anwendungsbezeichnung
		⇒ Länge = 5 Byte
	'52 61 6E 6B 6C'	⇒ Anwendungsbezeichnung = „Rankl“

Die FID im Dateibaum müssen so gewählt werden, dass die Dateien eindeutig selektiert werden können. Es ist also verboten, dass zwei EFs unter einem gemeinsamen DF den gleichen FID haben. Genauso wenig darf ein DF den gleichen FID wie ein direkt darunter befindliches EF haben, da dann das Betriebssystem entscheiden müsste, ob das Verzeichnis oder die Datei als Erstes selektiert wird.

Zum Finden eindeutiger FIDss eignen sich die drei folgenden Regeln:

- Regel 1: DFs und EFs innerhalb eines Verzeichnisses dürfen nicht den gleichen FID haben.
- Regel 2: Verschachtelte Verzeichnisse (d. h. DFs) dürfen nicht den gleichen FID haben.
- Regel 3: EFs innerhalb eines Verzeichnisses (d. h. MF oder DF) dürfen nicht den gleichen FID wie das über- oder untergeordnete Verzeichnis haben.

Short File Identifier (SFI)
Zur impliziten Selektion von Dateien unmittelbar in einem Kommando wird die Short-FID verwendet. Die Short-FID ist für ein EF optional, muss also nicht zwangsläufig vergeben werden. Sie wird bei der impliziten Dateiselektion im Kommando mit übergeben und hat deshalb nur eine Größe von 5 Bit. Die Short-FID kann somit Werte zwischen 1 und 30 annehmen, da eine Short-FID von '0' das aktuell selektierte EF adressiert.

DF Name
Die DFs sind die Zusammenfassungen von Dateien für die einzelnen Anwendungen. Sie sind in ihrer Art wie Verzeichnisse oder Ordner und können weitere Verzeichnisse oder EFs aufnehmen. In Zukunft könnte der dafür zur Verfügung stehende Adressraum mit dem 2 Byte langen FID zu klein werden. Deshalb besitzen DFs zusätzlich zu ihrem FID einen so genannten „DF Name".[4] Er hat nach ISO/IEC 7816-4 eine Länge zwischen 1 und 16 Byte. Der DF Name bietet einen genügend großen Adressraum, damit eine Chipkarten-Anwendung weltweit eindeutig ist. Da es bei freier Auswahl trotzdem irgendwann zwei DFs mit gleichem DF Name geben würde, wird der DF Name in der Regel nur in Verbindung mit dem in der ISO/IEC 7816-5 definierten AID (*application identifier*) benutzt. Der AID kann eine Länge zwischen 5 Byte und 16 Byte haben und setzt sich aus zwei von der ISO definierten Datenelementen zusammen. Der AID ist somit eine Teilmenge des DF Name.

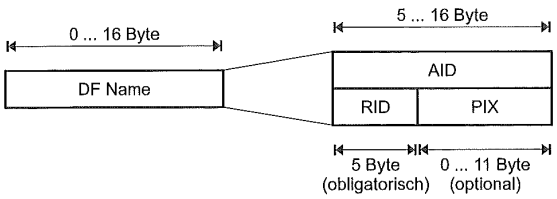


Bild 5.17 Der DF Name im Zusammenhang mit dem Aufbau des AID (application identifier) aus RID (registered identifier) und PIX (proprietary application identifier extension).

Aufbau und Codierung des Application Identifier (AID)
Der Application Identifier (AID) setzt sich selber wiederum aus zwei Datenelementen zusammen. Das erste Datenelement ist der Registered Identifier (RID) mit einer festen Länge von 5 Byte. Er wird entweder von einer nationalen oder internationalen Registrierungsstelle vergeben und beinhaltet einen Ländercode, eine Anwendungskategorie und eine Nummer für den Anwendungsanbieter. Dieser Zahlencode führt zu einer nur ein einziges Mal vergebenen RID, die weltweit zur Identifizierung einer bestimmten Anwendung benutzt werden kann.

Leider sind die Listen der vergebenen RIDs vertraulich, sodass sie nicht veröffentlicht werden können. Einige Beispiel veröffentlichter RID finden sich allerdings im Abschnitt „16.8 Ausgewählte RIDs". Anschriften der internationalen bzw. von nationalen Registrierungsstellen für RIDs finden sich im Anhang unter „15.7 Registrierungsstellen für RID".

[4] „DF Name" ist hier und auch im Folgendem ein englisches Wort, obwohl es identisch mit der deutschen Schreibweise ist.

Falls es notwendig ist, kann der Anwendungsanbieter der RID eine Proprietary Application Identifier Extension (PIX) nachstellen, die der optionale zweite Teil des AID ist. Die bis zu 11 Byte lange PIX kann zum Beispiel eine Serien- und Versionsnummer sein und damit zur Verwaltung der Anwendung benutzt werden.

Tabelle 5.8 Codierung des 5 Byte (= 10 Digits) langen Registered Identifier – RID

RID			Bedeutung
D1	**D2 . . . D4**	**D5 . . . D10**	
X	. . .	. . .	Kategorie der Registrierung 'A' – Internationale Registrierung 'D' – Nationale Registrierung
. . .	X	. . .	Ländercode, Codierung nach ISO 3166
. . .	. . .	X	Nummer des Anwendungsherausgebers, wird von der nationalen bzw. internationalen Registrierungsinstanz vergeben

Tabelle 5.9 Beispiel für eine national registrierte RID nach ISO/IEC 7816-5, anhand der RID von Wolfgang Rankl

RID			Bedeutung
D1	**D2 . . . D4**	**D5 . . . D10**	
'D'	. . .	. . .	die Kategorie der Registrierung ist national
. . .	'276'	. . .	der Ländercode der Registrierung nach ISO 3166 für Deutschland
. . .	. . .	'00 00 60'	von der nationalen Registrierungsinstanz vergebene Nummer des Anwendungsherausgebers

5.6.3 Selektion von Dateien

Die objektorientierten Dateiverwaltungssysteme erfordern die Selektion der Datei vor dem Zugriff. Damit wird dem Betriebssystem mitgeteilt, welche Datei von nun an angesprochen werden soll. Die erfolgreiche Selektion einer neuen Datei führt dazu, dass die bisherige Selektion ungültig wird. Deshalb kann immer nur eine Datei zur gleichen Zeit selektiert sein. Aufgrund der frei wählbaren FID müssen gewisse Einschränkungen der freien Adressierbarkeit von Dateien gemacht werden. Sonst würde sehr leicht der Fall eintreten, dass mehrere Dateien mit gleicher FID im Dateibaum zur Verfügung stehen und das Betriebssystem entscheiden müsste, welche Datei nun gemeint ist. Um diese Mehrdeutigkeiten zu vermeiden und dadurch auch unabhängig vom Suchalgorithmus der Dateiverwaltung des Betriebssystems zu sein, sind die Selektionsmöglichkeiten von Dateien limitiert.

Anders wäre es, wenn alle verwendeten FIDs im Dateibaum einzigartig wären. Dann wäre es ein Leichtes, über mehrere Verzeichnisgrenzen hinweg die gewünschte Datei auszuwählen. Doch genau dieser Fall kann nicht immer gewährleistet werden. Deshalb ist immer nur eine Auswahl innerhalb gewisser Grenzen möglich, da sonst die Eindeutigkeit der zu selektierenden Datei nicht mehr sichergestellt ist. Das MF kann jedoch von jedem Punkt innerhalb des Dateibaums selektiert werden, da seine FID im Dateibaum einzigartig ist. Die Auswahl von DFs der ersten Ebene unter dem MF ist beispielsweise nur von einem DF dieser Ebene oder vom MF aus möglich. Das Bild 5.18 zeigt als Muster einige mögliche und verbotene Selektionen.

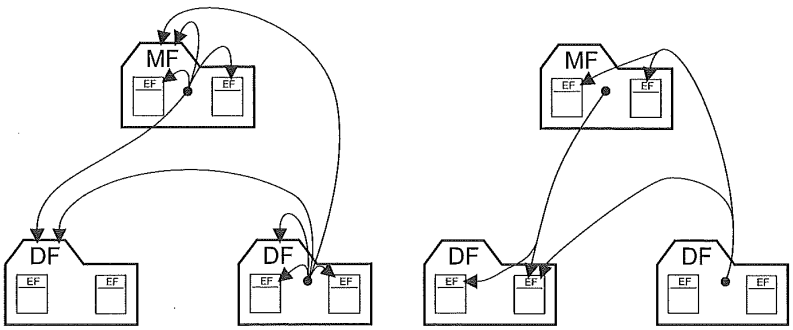


Bild 5.18 Beispiele für erlaubte Selektionsmöglichkeiten (links) und verbotene Selektionsmöglichkeiten bei Verwendung des FID bzw. DF Name (rechts). Es ist nur die direkte Selektion ohne Pfadangabe dargestellt.

Selektion von Verzeichnissen (MF, DF)
Das MF kann von jedem Ort innerhalb des Dateibaums entweder mit einer speziellen Option des Selektionskommandos oder unter Angabe seines nur einmal im Dateibaum vorkommenden FIDs '3F00' selektiert werden. Damit wird der Selektionszustand wiederhergestellt, wie er nach dem Reset der Chipkarte herrscht, da dort das MF implizit durch das Betriebssystem selektiert wurde. Die DFs können entweder mit ihrem FID oder dem DF Name, der die registrierte und somit einzigartige AID enthält, ausgewählt werden.

Explizite Selektion von EFs
Zur Auswahl von EFs stehen grundsätzlich zwei Methoden zur Verfügung: Bei der expliziten Selektion wird vor dem eigentlichen Zugriff ein eigenes Kommando (SELECT FILE) mit dem 2 Byte langen FID als Parameter zur Auswahl der gewünschten Datei zur Chipkarte gesendet. Danach kann auf die Datei mit allen weiteren Kommandos zugegriffen werden.

Implizite Selektion von EFs
Wird bei dem eigentlichen Dateizugriff in den Parametern eines Kommandos mit einem Short Identifier gleichzeitig die Datei ausgewählt, so spricht man von impliziter Selektion. Die implizite Selektion hat aber eine Reihe von Einschränkungen. Sie funktioniert nur bei EFs innerhalb des aktuell selektierten DFs oder MFs. Über Verzeichnisgrenzen hinweg ist somit keine Auswahl einer Datei möglich. Weiterhin ist sie nur mit bestimmten Zugriffskommandos möglich (z. B.: READ/UPDATE BINARY /RECORD), bei denen als Parameter der Short Identifier übergeben werden kann.

Der große Vorteil der impliziten Selektion liegt darin, dass mit einem einzigen Kommando selektiert und zugleich auf die Datei zugegriffen werden kann. Dies führt dazu, dass in vielen Fällen ein SELECT FILE-Kommando unnötig wird und es so zu einer Vereinfachung der Kommandoabläufe kommt. Aufgrund des reduzierten Kommunikationsbedarfs erreicht man mit einer impliziten Selektion von Dateien deutlich höhere Ausführungsgeschwindigkeiten.

Selektion durch Pfadangabe
Zusätzlich zur direkten Selektion existieren nach ISO/IEC 7816-4 noch zwei ergänzende Methoden der expliziten Selektion einer Datei mittels Pfadangabe. In der ersten Variante muss der Pfad von der aktuell selektierten Datei zur Zieldatei dem Betriebssystem übergeben werden. Die zweite Methode sieht vor, dass man den Pfad vom MF zur Zieldatei übergibt. Beide Methoden sind in vielen Chipkarten-Betriebssystemen realisiert und ermöglichen durch diese zusätzliche Funktionalität einen messbar verringerten Zeitbedarf bei der Abarbeitung von Kommandosequenzen.

5.6.4 Dateistrukturen von EFs

Die EFs bei Chipkarten haben im Gegensatz zu Dateien in Windows-Systemen eine interne Struktur. Diese Struktur kann je nach Verwendungszweck für jedes EF individuell gewählt werden, was große Vorteile für die äußere Welt mit sich bringt, denn durch diese internen Strukturen ist es möglich, Datenbestände so aufzubauen, dass sehr schnell und zielgerichtet auf sie zugegriffen werden kann.

Die Verwaltung dieser Datenstrukturen erfordert in der Chipkarte einen spürbaren Aufwand an Programmcode. Dies ist auch der Grund, warum die Datenstrukturen nicht alle symmetrisch zueinander sind, sondern nur in Varianten vorkommen, die in der Praxis häufig benötigt werden.

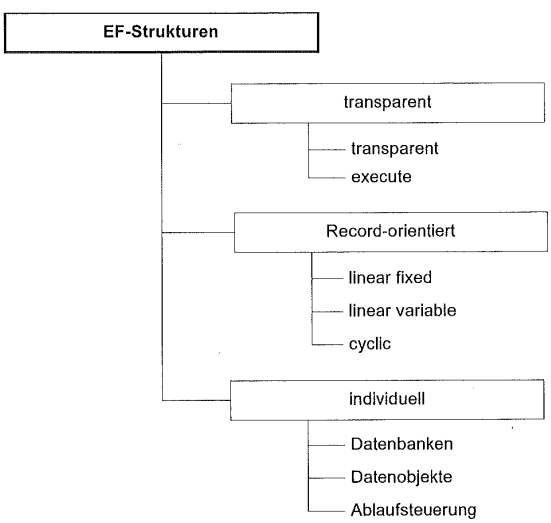


Bild 5.19 Klassifizierungsbaum der Dateistrukturen von EFs bei Chipkarten-Betriebssystemen.

Dateistruktur „transparent"
Die Dateistruktur „transparent" wird oft auch noch als binäre oder amorphe Struktur bezeichnet. Dies bedeutet in anderen Worten, dass eine transparente Datei keine innere Struktur aufweist. Auf die in der Datei enthaltenen Daten kann byteweise oder blockweise

schreibend oder lesend mit einem Offset zugegriffen werden. Dazu werden die Kommandos READ BINARY, WRITE BINARY und UPDATE BINARY verwendet.

Die minimale Größe eines EF mit der Struktur „transparent" ist 1 Byte. Eine maximale Größe ist in keiner Norm explizit festgelegt. Jedoch ist durch die maximale Anzahl von zu lesenden oder zu schreibenden Bytes im Short Format (255) bzw. im Extended Format (65536) und den maximalen Offset (32767) eine Größe von bis zu 65791 Byte bzw. 98303 Byte möglich. Diese Maximalangabe rückt bei den heutigen Chipkarten aufgrund der zur Verfügung stehenden Speichergrößen langsam in den Bereich des Möglichen, obwohl in der heutigen Praxis transparente Dateien selten größer als ein paar hundert Byte sind.

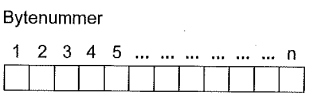


Bild 5.20 Der Aufbau der Dateistruktur „transparent".

Sollen zum Beispiel 6 Byte mit einem Offset von 4 Byte aus einer 12 Byte großen Datei gelesen werden, so findet der Zugriff folgendermaßen statt:

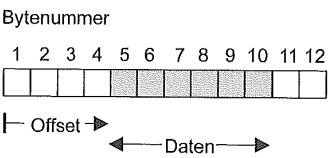


Bild 5.21 Beispiel für das Lesen von 6 Byte mit einem Offset von 4 Byte aus einer transparenten Datei.

Die Anwendung dieser Datenstruktur liegt hauptsächlich im Bereich von Daten, die nicht strukturiert aufgebaut oder sehr kurz sind. Ein typisches Einsatzgebiet wäre eine Datei mit einem digitalisierten Passfoto, das von einem Terminal aus der Chipkarte ausgelesen werden kann.

Mit der linear und eindimensional aufgebauten Datenstruktur lassen sich aber im Bedarfsfall auch alle anderen Strukturen simulieren. Allerdings gestaltet sich dann der Zugriff für das Terminal etwas aufwendiger, da intern in der Datei auch Parameter über den Aufbau abgespeichert werden müssen.

Dateistruktur „linear fixed"

Die linear fixed-Datenstruktur basiert auf der Verkettung von gleich langen Datensätzen, d. h. Records. Ein Record wiederum ist eine Aneinanderreihung einzelner Bytes. Auf die einzelnen Records dieser Dateistruktur kann wahlfrei zugegriffen werden. Die kleinste Zugriffsgröße ist ein einzelner Record, es ist also nicht möglich, nur auf Teile eines Records zuzugreifen. Gelesen bzw. geschrieben werden kann in diese Struktur mit READ RECORD, WRITE RECORD und UPDATE RECORD.

Der erste Record hat immer die Nummer 1. Die größte Recordnummer ist 'FE', also 254, da 'FF' für spätere Erweiterungen reserviert ist. Die Länge der einzelnen Records kann sich

aufgrund der Zugriffskommandos im Bereich von 1 bis 254 Byte bewegen, doch müssen alle Records die gleiche Länge aufweisen.

Die typische Anwendung dieser Dateistruktur ist ein Telefonverzeichnis, in dem als Erstes der Name steht und dann, ab einer festen Stelle, die dazugehörige Telefonnummer.

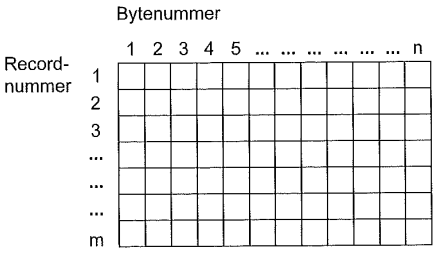


Bild 5.22 Der Aufbau der Dateistruktur linear fixed.

Dateistruktur „linear variable"

In der Dateistruktur „linear fixed" besitzen alle Records die gleiche Länge. Dadurch wird oft Speicherplatz verschwendet, da viele Record-orientierte Daten von variabler Länge sind. Man denke dabei nur an die Namen in einem Telefonverzeichnis. Dieser Forderung nach Speicherplatzminimierung kommt die Struktur linear variable nach. Hier kann jeder einzelne Record individuell eine definierte Länge annehmen. Dies führt zwangsläufig dazu, dass jeder Record ein zusätzliches Informationsfeld über seine Länge besitzt. Sonst ist der Aufbau analog dem in der Struktur „linear fixed".

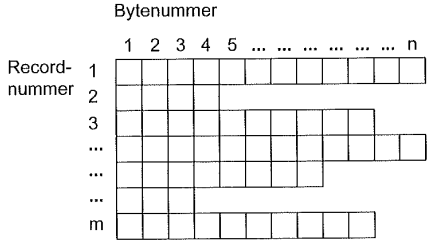


Bild 5.23 Der Aufbau der Dateistruktur linear variable.

Der erste Datensatz hat die Nummer 1, und es können maximal 254 Datensätze in einer Datei enthalten sein. Die Länge der einzelnen Records ist bestimmt durch die Zugriffskommandos von 1 Byte bis 254 Byte individuell einstellbar. Diese sind die gleichen wie bei der Datenstruktur „linear fixed", d. h. READ RECORD, WRITE RECORD und UPDATE RECORD.

Vorzugsweise findet diese Struktur dann Verwendung, wenn Datensätze mit sehr unterschiedlicher Länge gespeichert und Speicherplatz in der Chipkarte gespart werden soll. So könnte beispielsweise das obige Telefonverzeichnis dahingehend optimiert werden, dass die Records genauso lang sind wie die eigentlichen Einträge und somit nicht alle Records die gleiche Länge haben. Allerdings benötigt die Verwaltung dieser Dateistruktur im Chipkarten-Betriebssystem sowohl Programmcode als auch zusätzlichen Speicher für die indi-

viduellen Recordlängen. Deshalb bieten oftmals Betriebssysteme für Mikrocontroller mit wenig Speicherplatz diese Dateistruktur nicht an. Die ISO/IEC 7816-4 lässt diese Einschränkung konsequenterweise auch in einigen Profilen ausdrücklich zu.

Dateistruktur „cyclic"

Diese Struktur basiert auf der linear fixed-Datenstruktur. Sie besteht also aus einer bestimmten Anzahl von Records mit gleicher Länge. Zusätzlich beinhaltet das EF noch einen Zeiger, der immer auf den letzten geschriebenen Datensatz zeigt, welcher grundsätzlich die Nummer 1 hat. Erreicht der Zeiger den letzten Datensatz im EF, wird er beim nächsten Schreibzugriff vom Betriebssystem automatisch wieder auf den ersten Datensatz gesetzt. Er verhält sich also wie bei einer analogen Uhr mit einem Stundenzeiger.

Enthält die cyclic-Datei n Records, so hat der letztgeschriebene die Nummer 1. Der vorhergehend geschriebene Record hat die Nummer 2 und der „älteste" Record die Nummer n. Man kann auch auf diese Dateistruktur, wie im Übrigen auf alle der drei Record-orientierten Dateistrukturen, mit Adressierung des ersten (*first*), letzten (*last*), vorherigen (*previous*) und nächsten (*next*) Record darauf zugreifen.[5]

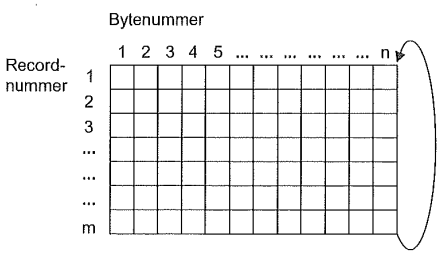


Bild 5.24 Der Aufbau der Dateistruktur „cyclic".

Die Anzahl und Länge der Datensätze ist völlig analog dem bei „linear fixed". Es können also aufgrund der Restriktionen durch die Schreib- und Lesekommandos maximal 254 Datensätze mit einer Maximallänge von je 254 Bytes erzeugt werden.

Die typische Anwendung für diese Struktur sind Dateien innerhalb der Chipkarte für die Aufzeichnung von Protokollen, da der jeweils älteste Eintrag immer durch den neuesten überschrieben wird.

Dateistruktur „execute"

Die folgende Dateistruktur ist im Prinzip nicht eigenständig, da sie auf der transparenten Datenstruktur beruht. Sie ist in der europäischen Norm EN 726-3 beschrieben und bietet innerhalb des Betriebssystems große Erweiterungsmöglichkeiten. Die Datenstruktur „execute" ist nicht zum Speichern von Daten, sondern für die Ablage von ausführbarem Programmcode gedacht.[6] Zugreifen kann man auf Dateien mit der Struktur „execute" mit den entsprechenden Kommandos für transparente Dateistrukturen. Allerdings schafft man damit eine Hintertür, denn jeder, der auf diese Datei schreiben kann, hat dadurch die Möglichkeit, eigenen Programmcode inklusive trojanischer Pferde in die Karte zu laden.

[5] Siehe auch Abschnitt 7.2 „Schreib- und Lesekommandos".
[6] Siehe auch Abschnitt 5.12 „Nachladbarer Programmcode".

Die maximale Programmgröße ist analog der maximalen Größe einer transparenten Datei und beträgt 65791 Byte bzw. 98303 Byte (ohne/mit Adressierung über Offset bei UPDATE BINARY). Eine eigene innere Struktur, wie etwa bei den Record-orientierten Dateien, ist nicht festgelegt. Sie kann aber durch den in der Datei enthaltenen Programmcode durchaus eingeführt werden, sodass das ausführbare Programm eigene Datenbereiche in der execute Datei anlegt und intern darauf zugreift.

Dateistruktur für Datenbanken, Database File
Die ISO/IEC 7816-7 definiert mit SCQL eine Untermenge von SQL für Chipkarten. Um die Daten in einem für die SCQL-Kommandos auslesbaren Form im Dateisystem der Chipkarte ablegen zu können, ist es notwendig, eine eigene Dateistruktur dafür vorzusehen. Diese ist in ihrem Aufbau nicht normiert, sondern dem jeweiligen Betriebssystemhersteller überlassen. In dem Database File sind die eigentlichen Nutzdaten der Datenbank, unterschiedliche Sichten auf die Datenbank, die Zugriffsrechte auf die Datenbank und die Benutzer-Profile abgelegt.

Dateistruktur für Datenobjekte
Die beiden Kommandos GET DATA und PUT DATA der ISO/IEC 7816-4 werden benutzt, um TLV-codierte Datenobjekte in der Chipkarte abzulegen und von dort wieder auszulesen. Dies kann entweder völlig unabhängig von der Dateiverwaltung realisiert werden oder innerhalb einer Dateiverwaltung durch eine besondere Dateistruktur für die Ablage von Datenobjekten. Findet eine Realisierung über die Dateiverwaltung statt, kann man beispielsweise eine geringfügig angepasste transparente oder linear variable Dateistruktur als Aufbewahrungsplatz für die Datenobjekte benutzen. Die Kommandos GET DATA und PUT DATA greifen dann über die Dateiverwaltung auf diese modifizierten Dateistrukturen zu.

Dateistruktur für Ablaufsteuerung
Weist ein Chipkarten-Betriebssystem eine Ablaufsteuerung für Kommandosequenzen auf, dann müssen die Informationen, wann welche Kommandos akzeptiert werden, ebenfalls im Speicher abgelegt werden. Dies geschieht üblicherweise in einer Datei mit besonders angepasster Struktur für diese Aufgabe. Diese ist jedoch nicht genormt, und jedes Betriebssystem mit Ablaufsteuerung besitzt ausnahmslos ein eigenes und nicht zu anderen kompatibles Format.

5.6.5 Zugriffsbedingungen auf Dateien

Alle Dateien besitzen im Rahmen ihres objektorientierten Aufbaus Informationen, die den Zugriff im Rahmen des Dateimanagements regeln. Physikalisch ist die dazugehörige Codierung immer im Header einer Datei untergebracht. Die gesamte Sicherheit der Dateiverwaltung einer Chipkarte basiert auf der Rechteverwaltung für den Dateizugriff, da bei ihr der Zugriff auf die Dateiinhalte geregelt wird.

Die Zugriffsbedingungen werden bei der Erzeugung einer Datei festgelegt und sind dann im Regelfall nicht mehr zu verändern. Die für eine Datei möglichen Zugriffsbedingungen differieren je nach den im Betriebssystem vorhandenen Kommandos sehr stark. Es hat beispielsweise keinen Sinn, Zugriffsbedingungen für READ RECORD zu definieren, wenn dieses Kommando in einem Chipkarten-Betriebssystem nicht vorhanden ist.

Beim MF und den DFs werden im Gegensatz zu den EFs keine Informationen über den Datenzugriff (Schreib- oder Leserechte) gespeichert, sondern unter anderem die Zugriffsbedingungen für die Erzeugung von neuen Dateien. Je nach Dateityp sind also verschiedene Zugriffsbedingungen gespeichert: In den EFs für den Zugriff auf die Dateiinhalte und im MF und den DFs Bedingungen, die innerhalb dieser Organisationsstruktur gelten.

Bei der Festlegung der Zugriffsbedingungen kann zwischen zustands- und kommandoorientierten Zugriffsbedingungen unterschieden werden. Bei den zustandsorientierten Zugriffsbedingungen wird der aktuelle Sicherheitszustand verknüpft mit einem definierbaren logischen Vergleich mit der entsprechenden Zugriffsbedingung der Datei verglichen. Bei den aktuellen Sicherheitszuständen gibt es die Variante globaler Sicherheitszustand und lokaler Sicherheitszustand. Der globale Sicherheitszustand ist der Sicherheitszustand des MFs, d. h. der Chipkarte als Ganzes. Der lokale Sicherheitszustand ist der Zustand des aktuell selektierten Verzeichnisses, d. h. des DFs, oder des darüber befindlichen Verzeichnisses. Durch erfolgreiche Ausführung von Identifizierungs- und Authentisierungskommandos mit bestimmten Schlüsseln kann sowohl der globale, als auch der lokale Sicherheitszustand geändert werden. Beim Zugriff auf eine Datei wird nun dieser aktuelle Sicherheitszustand mit dem in der Datei als Zugriffsbedingung festgelegten Zustand durch eine logische Vergleichsoperation verglichen. Ist der Vergleich erfolgreich, so darf auf die Datei zugegriffen werden. So kann zum Beispiel ein Lesezugriff ab dem Zustand 5 erlaubt sein. Für jeden Zustand, der kleiner ist als 5, wäre dann der Lesezugriff verboten. Natürlich ist es auch möglich, für einen Zugriff mehrere unterschiedliche Zustände festzulegen. So könnte ein Lesezugriff im Zustand 5, 8 und 9 erlaubt sein.

In der Regel wird bei einem erfolgreichen Vergleich der Fehlbedienungszähler des jeweiligen Geheimnis (*error counter*) wieder auf null zurückgesetzt. Beim nicht erfolgreichen Zugriff wird der entsprechende Fehlbedienungszähler erhöht; falls er seinen Maximalwert erreicht hat, wird der entsprechende Schlüssel gesperrt. Die zustandsorientierten Zugriffsbedingungen mögen auf den ersten Blick relativ kompliziert erscheinen. Sie lassen jedoch bei der Gestaltung von Anwendungen einen enormen Spielraum zu und können im Prinzip jede mögliche Architektur von Anwendungen abdecken. Der Nachteil ist, dass sie verhältnismäßig komplex sind.

Im Gegensatz dazu werden bei kommandoorientierten Zugriffsbedingungen die vor dem Zugriff korrekt auszuführenden Kommandos definiert. Dies betrifft vor allem Authentisierungs- und Identifizierungskommandos. Kommandoorientierte Zugriffsbedingungen sind in der Chipkartenwelt weit verbreitet, ihr bekanntester Vertreter ist die Chipkarte für GSM. Bei kommandoorientierten Zugriffsbedingungen enthält die Zugriffstabelle in der Datei Informationen über die erfolgreich auszuführenden Kommandos für jede der Zugriffsarten. Oftmals werden die Kommandos noch nach spezifischen Schlüsseln aufgeteilt. In der Praxis würde beispielsweise eine bestimmte Datei als Lesebedingung die vorhergehende Identifizierung des Benutzers mit dem Kommando „VERIFY“ und seiner PIN Nummer eins erfordern. Erst nach erfolgreicher Abarbeitung dieses Kommandos kann die Datei gelesen werden. Der Vorteil dieser Art von Zugriffsbedingung ist der einfache Aufbau, mit dem in aller Regel die meisten Anwendungsfälle abgedeckt werden können. Gerade aber für Multiapplikations-Chipkarten sind kommandoorientierte Zugriffsbedingungen oftmals nur mit zusätzlichem Aufwand einsetzbar. Bei Betriebssystemen mit nachladbarem Programmcode

würde dies beispielsweise auch eine Erweiterung der Zugriffstabelle in den Dateien erfordern, da dort explizit auf bestimmte Kommandos referiert werden muss. Dies macht diese Art der Zugriffsbedingungen in manchen Fällen etwas unflexibel.

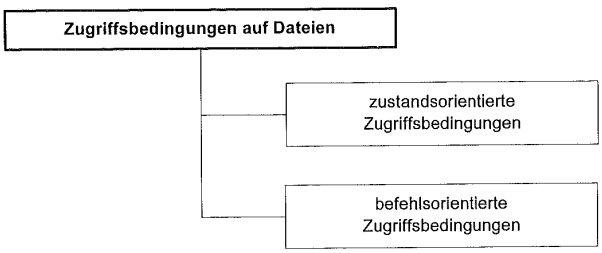


Bild 5.25 Klassifizierungsbaum der beiden möglichen Arten von Zugriffsbedingungen auf Dateien.

Alle möglichen Zugriffsarten auf ein EF müssen durch Rechte genau geregelt sein. Dies können je nach Betriebssystem mehr oder weniger viele Kommandos sein. Im Überblick seien deshalb hier einige der häufigsten Zugriffskommandos auf Dateien genannt:

APPEND	Vergrößern einer Datei
DELETE FILE	Löschen einer Datei
INCREASE / DECREASE	Berechnungen innerhalb einer Datei
INVALIDATE	Blockieren einer Datei
LOCK	Endgültiges Sperren einer Datei
READ / SEEK	Lesen/Suchen in einer Datei
REHABILITATE	Entblocken einer Datei
WRITE / UPDATE	Schreiben in eine Datei

Die Zugriffsbedingungen in einem DF unterscheiden sich grundlegend von denen eines EFs. Sie geben die Bedingungen an, damit innerhalb des betreffenden Verzeichnisses bestimmte Kommandos ausgeführt werden dürfen. Im Folgenden die drei wichtigsten Zugriffskommandos im Überblick:

CREATE	Erzeugen einer neuen Datei
DELETE FILE	Löschen einer Datei
REGISTER	Registrieren einer neuen Datei

Zustandsorientierte Zugriffsbedingungen werden vor allem in multifunktionalen Chipkarten-Betriebssystemen wie STARCOS eingesetzt, da sie sehr flexibel und anpassungsfähig sind. Die meisten großen Chipkarten-Anwendungen, wie beispielsweise GSM, ec-Karte mit Chip in Deutschland, regeln die Dateizugriffe jedoch über kommandoorientierte Zugriffsbedingungen. Dieses Zugriffsverfahren benötigt geringfügig weniger Programmcode und Datenspeicher als die zugriffsorientierte Variante. Allerdings erkauft man sich dies durch eine etwas geringere Flexibilität. Beide Verfahren haben ihre kleinen Vor- und Nachteile, und Diskussionen für oder gegen eine der beiden Zugriffsmethoden enden meist in philosophischen Fragen zum Betriebssystemdesign. Nach ISO/IEC 7816-9 wurde aber mittlerweile ein internationaler Standard festgelegt,

mit dem in allgemeiner Form Zugriffsbedingungen auf Ressourcen (d. h. auch auf Dateien) flexibel gestaltet werden kann. Weiter unten findet sich eine Erläuterung dieses sehr mächtigen Konzepts.

5.6.6 Attribute von Dateien

Innerhalb der objektorientierten Definition besitzen alle EFs spezielle Attribute, um noch zusätzliche Eigenschaften der Datei zu definieren. Dies ist jedoch abhängig vom Betriebssystem und auch vom Einsatzgebiet der Chipkarte. Die Attribute definieren Eigenschaften von EFs, die sich meist auf das Speichermedium EEPROM beziehen. Die Ursache sind die potenzielle Unsicherheit von Datenerhalt und mögliche Schreibfehler bei EEPROM-Operationen. Die Attribute werden beim Erzeugen einer Datei definiert und können dann meist nicht mehr verändert werden.

Attribut für WORM
Ein gleichfalls auf dem Speichermedium EEPROM basierendes Attribut nennt sich WORM (*write once, read multiple*). Besitzt eine Datei ein solches Attribut, so können Daten nur ein einziges Mal in die Datei geschrieben werden. Lesen lassen sich diese Daten aber unbegrenzt oft. Diese Eigenschaft kann entweder von der Hardware des EEPROMs unterstützt werden oder ist als Softwarefunktion realisiert. Verwendung findet das WORM-Attribut beispielsweise zum einmaligen und irreversiblen Schreiben einer Seriennummer in eine Datei. Andere Anwendungen dieser Dateieigenschaft finden sich im Bereich der Personalisierung, in dem nicht mehr änderbare Daten (z. B. Name, Verfallsdatum) in die Karte unwiderruflich geschrieben werden.

Der Sinn der Verwendung dieses Attributes besteht im Schutz von sensiblen Daten vor Überschreiben. Optimal ist der Schutz natürlich erst dann, wenn auf Ebene der Hardware ein WORM-Zugriff möglich ist. Das heißt, wenn das EEPROM einen Hardwareschutz hat und nur ein einmaliges Schreiben zulässt. Doch selbst eine Realisierung in Software ermöglicht einen viel besseren Schutz als vergleichbare Mechanismen.

Attribut für häufiges Schreiben (*high update activity*)
Ein vor allem im GSM-Bereich definiertes und verwendetes Attribut ist ein Kennzeichen für „high update activity". Es existiert nur aufgrund der begrenzten Anzahl von Schreib-/Löschzyklen auf das EEPROM. Dateien mit diesem Attribut können sehr oft geschrieben werden, ohne dass Schreibfehler eine Auswirkung auf den Datenerhalt haben. Erreicht werden kann dies durch eine Mehrfachablage beim Schreiben der Daten und eine Mehrheitsentscheidung beim Lesen. Üblich ist dabei ein dreifach paralleles Abspeichern der Daten beim Schreiben und einer 2-aus-3-Mehrheitsentscheidung beim Lesen. Alternativ zu diesem Mechanismus kann auch eine im Schreibfehlerfall oder bei auftretenden Prüfsummenfehler nach außen hin transparente Umschaltung der mehrfach vorhandenen Datenfelder realisiert werden.

Attribut für EDC-Benutzung
Ein für besonders sensible Daten benutztes Attribut ist ein spezieller Schutz der Nutzdaten in der Datei mit einem EDC (*error detection code*). Damit kann das Umkippen von Speicherbits im EEPROM zumindest erkannt werden. Mit einer zusätzlichen Mehrfachablage

und EDC-Schutz würde sich auch eine Fehlerkorrektur bei gekippten Speicherzellen vornehmen lassen. Diese ECC (*error correction code*) -Eigenschaft wird vornehmlich im Bereich von elektronischen Geldbörsen angewandt. Dort bedeutet ein Kippen von Speicherzellen den realen Verlust von Geld, da in der Datei der aktuelle Börseninhalt gespeichert ist. Um die Auswirkungen des Kippens von Zellen zu minimieren, verwendet man ebendarum diese EDC- bzw. ECC-Attribute bei Dateien.

Attribut für Atomare Schreibzugriffe
In neueren Betriebssystemen für Chipkarten ist oft ein Mechanismus integriert, der Sicherstellt, dass eine Datei bei einem Schreibzugriff entweder vollständig oder gar nicht geschrieben wird.[7] Da dieser Mechanismus die Zugriffszeit beim Schreiben in eine Datei mehr als verdoppelt, sollte er nicht prinzipiell bei allen Dateien angewendet werden. Um selektiv für jede Datei den Schreibmechanismus bestimmen zu können, gibt es ein spezielles Attribut dafür.

Attribut für gleichzeitigen Zugriff
Chipkarten-Betriebssysteme, die mehrere logische Kanäle unterstützen, haben oftmals ein spezielles Dateiattribut für gleichzeitigen Dateizugriff. Dabei wird für eine Datei explizit erlaubt, dass Kommandos, die die Chipkarte über unterschiedliche und parallel geöffnete logische Kanäle erhält, auf eine Datei lesend oder schreibend zugreifen können. Es ist wichtig, dass diese Eigenschaft für eine Datei besonders gekennzeichnet ist, da bei einem parallelen Zugriff über unterschiedliche Kanäle Daten von einem Kanal verändert werden können, während sie unmittelbar vorher oder nachher über den anderen Kanal gelesen werden. Falls beide Prozesse nicht synchronisiert sind, werden je nach zeitlichem Eintreffen der Kommandos an die Chipkarte unterschiedliche Dateiinhalte ausgelesen. Deshalb sind in der Regel diese gleichzeitigen Zugriffe grundsätzlich untersagt. Nach der Selektion einer Datei wird der Zugriff über alle anderen Kanäle darauf temporär gesperrt. Erst nach einer Deselektion ist es wieder möglich, von einem anderen logischen Kanal darauf zuzugreifen. Das Attribut für gleichzeitigen Zugriff schaltet diese Sperre für eine bestimmte Datei aus. Für die Synchronisation von parallelen Schreib- und Leseprozessen sind dann jedoch die entsprechenden Anwendungen im Terminal verantwortlich. Greifen diese jedoch auf die Datei nur lesend zu, tritt das Problem natürlich nicht auf.

Attribut für Auswahl der Datenübertragung
Die Dateiverwaltungen von Chipkarten, die sowohl mit einer kontaktbehafteten als auch mit einer kontaktlosen Schnittstelle ausgerüstet sind, besitzen manchmal ein Dateiattribut, das regelt, welche der beiden Schnittstellen für den Zugriff auf eine Datei verwendet werden darf. Damit lässt sich für jede Datei individuell regeln, ob der Zugriff mit Kommandos über die kontaktlose und/oder kontaktbehaftete Schnittstelle erfolgen darf. Damit ist es beispielsweise bei einer elektronischen Geldbörse sehr einfach, das Bezahlen nur über die kontaktlose Datenübertragung und das Aufladen nur über die kontaktbehaftete Schnittstelle zuzulassen.

[7] Siehe auch Abschnitt 5.10 „Atomare Abläufe".

5.7 Dateiverwaltung

Alle Dateien einer Chipkarte sind im EEPROM gespeichert. Dies ist die einzige Speicherart in einer Chipkarte, in der Daten auch ohne Stromversorgung gespeichert bleiben und zusätzlich bei Bedarf und vorhandener Stromversorgung wieder geändert werden können. Es ist auch die einzige Möglichkeit, Informationen zwischen zwei Sitzungen hinweg zu erhalten, da das RAM nach der Deaktivierung der Chipkarte seinen Inhalt verliert und der Inhalt des ROM nach der Halbleiterfertigung nicht mehr geändert werden kann.

In früheren Chipkarten wurde auf die Dateien direkt mit physikalischen Adressen zugegriffen. Eigentlich waren dies im engeren Sinn keine Dateien, sondern der gesamte Speicher war linear von außen adressierbar und konnte mit Schreib- und Lesekommandos angesprochen werden. Dies ist jedoch in modernen Betriebssystemen aus Sicherheits- und Anwendungsgründen nicht mehr vorgesehen. Standard sind momentan objektorientierte Dateiverwaltungen mit Informationen über die Zugriffsbedingungen direkt an der Datei. Die Verwaltung und Organisation dieser Dateien ist die Aufgabe des Betriebssystemteils „Dateiverwaltung".

Jede Datei muss bei einem objektorientierten Aufbau somit über einen Dateideskriptor verfügen, der alle entsprechenden Informationen über die Datei selber enthält. Der Dateideskriptor wird in der Chipkartentechnik auch als Header einer Datei bezeichnet. Die Dateninhalte, d. h. die Nutzdaten, befinden sich im so genannten „Body" der Datei.

Die Informationen des Dateideskriptors sind stark abhängig von den Möglichkeiten, die die Dateiverwaltung bietet. Folgendes muss jedoch in jedem Fall im Dateideskriptor enthalten sein:

- Name der Datei (z. B.: FID = '0001')
- Typ der Datei (z. B.: EF)
- Struktur der Datei (z. B.: linear fixed)
- Größe der Datei (z. B.: 3 Records à 5 Byte)
- Zugriffsbedingungen (z. B.: READ = nach PIN-Eingabe)
- Attribute (z. B.: WORM)
- Verbindung zum Dateibaum (z. B.: direkt unter dem MF)

Der Name der Datei ist im Falle eines EFs oder des MFs der 2 Byte lange FID (*file identifier*). Ist die Datei ein DF, ist zusätzlich noch ein AID (*application identifier*) Bestandteil des Dateideskriptors. Der Typ der Datei, also EF, DF oder MF, muss ebenfalls angezeigt sein.

Abhängig vom Typ existiert ein Datenelement im Header, das die interne Struktur (*transparent, linear fixed, linear variable, cyclic, executable*) beschreibt. Von der Struktur wiederum beeinflusst sind alle Informationen über die Länge des transparenten Datenteils oder die Anzahl und Länge der Records.

Nachdem nun so alle grundlegenden Eigenschaften der Datei beschrieben sind, benötigt das Betriebssystem noch detaillierte Angaben über die Zugriffsbedingungen, d. h. in welchem Zustand welche Kommandos wie zugreifen dürfen. Diese müssen für jedes mögliche Kommando gesondert spezifiziert sein. Falls es die Dateiverwaltung unterstützt, können noch spezielle Attribute, wie high update activity, WORM oder EDC Schutz, indiziert sein.

All diese vorstehenden Informationen betreffen die Datei als Objekt selber. Um die Position im Dateibaum festzulegen, bedarf es noch einiger Zeiger, mit deren Hilfe die genaue Lage innerhalb des MFs oder des DFs spezifiziert ist.

Zeiger-orientierte Dateiverwaltung

In einfachen Chipkarten-Betriebssystemen haben die Dateiheader in Abhängigkeit des Dateityps (MF, DF, EF) eine feste Länge. Dies reduziert den Verwaltungs- und Berechnungsaufwand für die interne Dateiverwaltung. Allerdings hat so ein Aufbau den großen Nachteil, dass er gegenüber Erweiterungen vergleichsweise unflexibel ist. Auch kann

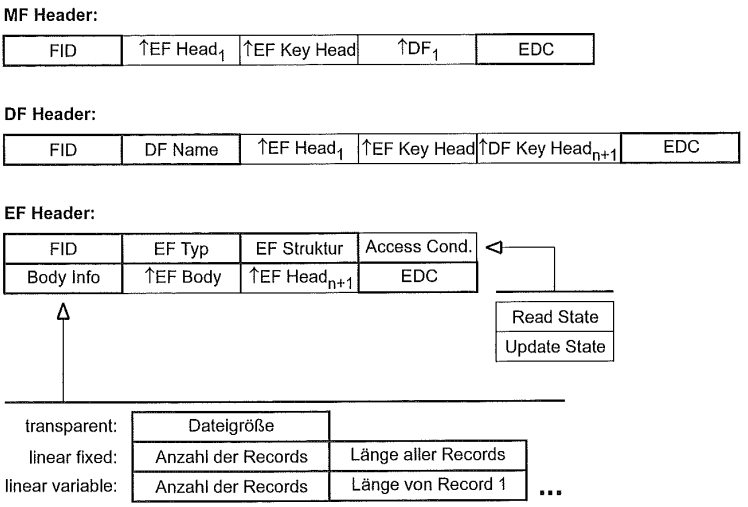


Bild 5.26 Möglicher Aufbau der Datei-Header für MF, DF und EF (internal und working) in einer Zeiger-orientierten Dateiverwaltung. Die breiten Rahmen kennzeichnen TLV-codierte Datenobjekte, die obligatorisch vorhanden sein müssen, und die schmalen Rahmen Datenobjekte, die optional sind. Die Nummerierung ist nur innerhalb des jeweiligen Verzeichnisses gültig und nicht global über alle Dateien. Die Grundlage sind die bei Small-OS festgelegten Anforderungen an eine einfache Dateiverwaltung. Folgende Abkürzungen wurden benutzt: „Access Cond." für Zugriffsbedingung und „Head" für Header.

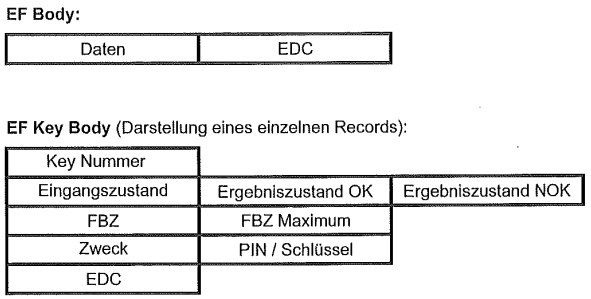


Bild 5.27 Überblick über einen möglichen Aufbau der Datei-Bodies für internal und working EFs in einem Chipkarten-Dateiverwaltungssystem. Alle TLV-codierten Datenobjekte müssen obligatorisch vorhanden sein. Die Grundlage sind die bei Small-OS festgelegten Anforderungen an eine einfache Dateiverwaltung. FBZ ist die Abkürzung für Fehlbedienungszähler und EDC (*error detection code*) für Fehlererkennungscode.

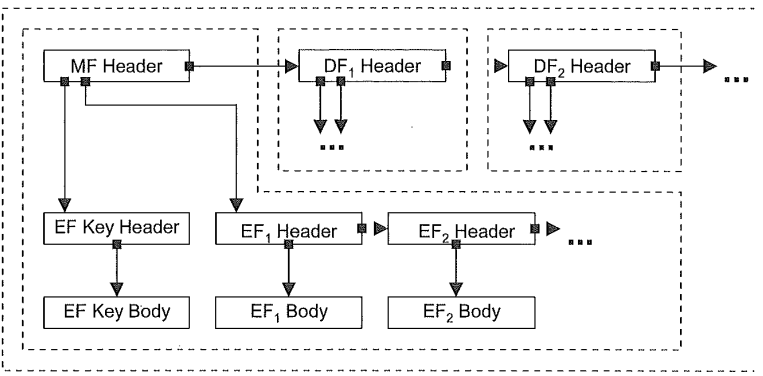


Bild 5.28 Überblick über einen möglichen Aufbau der Zeiger- und Datenstrukturen in einem Chipkarten-Dateiverwaltungssystem. Die Grundlage dafür sind die bei Small-OS festgelegten Anforderungen an eine einfache Dateiverwaltung. Die gestrichelt eingegrenzten Bereiche symbolisieren den für das jeweilige Verzeichnis zur Verfügung stehenden Speicher.

damit beispielsweise keine unbegrenzte oder auch nur sehr große Anzahl von unterschiedlichen Zugriffsbedingungen pro EF verwirklicht werden, da der prophylaktisch für die Speicherung der Zugriffsbedingungen im Header zur Verfügung zu stellende Speicher in den meisten Anwendungsfällen nicht benutzt würde. Dies ist der Grund, warum variable lange Dateiheader bei Dateiverwaltungen in Chipkarten sehr beliebt sind. Diese variablen Header können dann vom Chipkarten-Betriebssystem je nach Anwendung und daraus resultierenden Anforderungen an die im Header zu speichernden Information automatisch angepasst werden.

FAT-orientierte Dateiverwaltung (*file allocation table*)

Eine seit vielen Jahren auf den Festplatten von PCs verbreitete Art der Dateiverwaltung basiert auf File Allocation Tables (FAT). Dieses Verfahren kann ohne wesentliche Änderungen für die Speicherverwaltung von Chipkarten übertragen werden. Dazu wird das zu verwaltende EEPROM in viele gleich große Stücke, so genannte Sektoren, aufgeteilt. Idealerweise entspricht sowohl die Größe eines Sektors als auch seine Startadresse dabei der einer EEPROM-Page. Damit ist es möglich auf der Basis von ganzen EEPROM-Speicherseiten zu schreiben und zu löschen.

Die FAT enthält auf jeden Sektor des Speichers einen Zeiger, wobei die FAT-Einträge untereinander ebenfalls durch Zeiger miteinander verknüpft sind. Nicht belegte FAT-Einträge und fehlerhafte Sektoren sind mit einem besonderen Eintrag gekennzeichnet. Die FAT ließe sich noch hinsichtlich Speicherbedarf erheblich komprimieren, wenn man eine direkte Zuordnung von FAT-Eintrag und Sektor einführt. In diesem Fall würden die Zeiger innerhalb der FAT überflüssig.

Bild 5.29 zeigt eine mögliche Realisierungsform einer FAT zur Dateiverwaltung in einer Chipkarte. Der Dateidescriptor mit den wesentlichen Informationen über den Aufbau der Datei weist mit einem Zeiger auf den Starteintrag in der FAT. Dort sind durch FAT-interne Zeiger eine der Dateigröße entsprechende Anzahl von Sektoren verknüpft, wobei der letzte FAT-Eintrag die EOF-Kennung (*end of file*) ist. Von der FAT aus wird eine 1:1-Beziehung

zu entsprechenden Sektoren im Speicher der Chipkarte aufgebaut, in denen sich dann die zum Dateidescriptor gehörenden Nutzdaten befinden.

Ob Dateiverwaltungen durch Zeiger verknüpfte Header und Bodies oder durch ein FAT realisiert werden, hängt zum großen Teil von diversen technischen Rahmenbedingungen ab. Beide Lösungsansätze haben sowohl Vor- als auch Nachteile. FAT-orientierte Dateisysteme benötigen zum einen für die FAT selber und gerade bei kleinen Dateien gegenüber Zeiger-orientierten Dateisystemen überproportional viel Speicher. Andererseits entfällt bei FAT-orientierten Dateisystemen eine Speicherdefragmentierung, da es zu dieser vom Grundsatz her überhaupt nicht kommen kann.

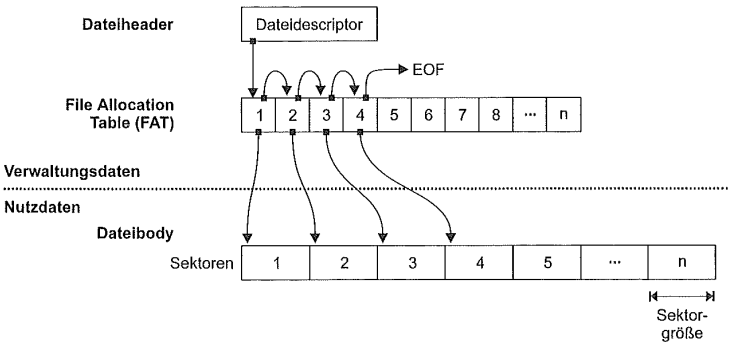


Bild 5.29 Grundlegender Aufbau einer FAT (*file allocation table*) zur Verwaltung des Dateispeichers in einer Chipkarte. Das Ende der Nutzdaten wird in der FAT durch ein EOF (*end of file*) gekennzeichnet.

Die Anforderung und Freigabe von Speicher im Speichermanagement eines Chipkarten-Betriebssystems wird über verschiedene Dienste realisiert. Diese sind auf unterster Ebene: Speicher anfordern, Speicher freigeben, Speicher vergrößern, Daten lesen, Daten schreiben und Daten atomar schreiben. Auf der darauf aufbauenden Schnittstelle des Dateimanagements werden üblicherweise die Dienste Datei anlegen, Datei löschen, Daten aus Datei lesen, Daten in Datei schreiben, MF selektieren, übergeordnetes DF selektieren, Datei mit FID selektieren und Datei mit DF Name selektieren realisiert.

Aufteilung des Speichers in Speicherseiten

Ein generelles Problem der gesamten Dateiverwaltung sind die limitierten Schreib-/Löschzyklen des EEPROM und die Aufteilung in einzelne Speicherseiten. Dies beeinflusst den gesamten Aufbau der Dateiverwaltung und der internen Dateistrukturen erheblich. Sowohl die Header als auch die Bodies der Dateien müssen in ihrer Größe an die vorgegebene Größe der Speicherseiten angepasst sein, sodass es nicht zu Überschneidungen zwischen unterschiedlichen Speicherseiten kommen kann. Auch müssen die Informationen über die Verwaltung von Dateien streng von den eigentlichen Dateninhalten im Speicher getrennt sein. Wäre dies nicht der Fall, könnte es so zu ungewollten Seiteneffekten zwischen den Verwaltungsdaten im Header und den Nutzdaten im Body kommen. Damit könnte die gesamte Sicherheitsstruktur innerhalb des Chipkarten-Betriebssystems zerstört werden. Dies sei an dem folgendem Beispiel kurz verdeutlicht:

Angenommen, die Zugriffsbedingungen für eine Datei mit geheimen und nicht auslesbaren Schlüsseln seien auf derselben Speicherseite abgelegt wie die öffentlich schreibbaren Nutzdaten einer anderen Datei. Würde nun der Schreibvorgang in die Datei abgebrochen, z. B. durch Herausziehen der Karte aus dem Terminal, so würde dies die auf der gleichen Speicherseite abgelegten Zugriffsbedingungen beeinflussen. Im ungünstigsten Fall wären nach diesem Vorgang keine Bedingungen für den Zugriff mehr vorhanden und die Datei mit den geheimen Schlüsseln somit von jedermann lesbar. Deshalb ist es von elementarer Bedeutung, dass die internen Dateistrukturen für Verwaltungs- und Nutzdaten jeweils auf einzelne Speicherseiten aufgeteilt sind.

Separierung von DFs

Die Funktion eines DFs kann auch so interpretiert werden, dass es den für eine bestimmte Anwendung zur Verfügung stehenden Speicher darstellt. Innerhalb dieses Bereiches ist der Anwendungsbetreiber vollständig für seine Anwendung verantwortlich und kann dort im Wesentlichen tun und lassen, was er will. Er darf aber keinesfalls die Möglichkeit haben, in irgendeiner Form über seine Anwendung hinaus in eine andere Anwendung hinein zu adressieren und dort Daten zu lesen oder zu verändern. Deshalb haben einige Chipkarten-Betriebssysteme besondere Mechanismen, die grundsätzlich bei jedem Speicherzugriff eine Prüfung vornehmen, ob sich die physikalische Adresse innerhalb des aktuellen DFs befindet. Ist dies nicht der Fall, wird sofort mit einem schweren internen Fehler abgebrochen.

Die Überwachung dieser Adressen findet heute aufgrund fehlender Hardware-Unterstützung durch entsprechende Programmroutinen im Betriebssystem statt. Dies ist natürlich eine sicherheitstechnisch wesentlich weichere, da leichter zu umgehende Lösung, als man es mit einer entsprechenden Hardware erreichen könnte. In Zukunft werden jedoch wohl auch die Chipkarten-Mikrocontroller, analog zu allen aktuellen CPUs, eine so genannte MMU (*memory management unit*) besitzen.[8] Diese kann dann zur sicheren Kontrolle der Speicherzugriffe innerhalb eines DFs genutzt werden. Bis dahin gibt es nur die Möglichkeit, die Adressgrenzen der DFs durch entsprechende Betriebssystemroutinen zu überwachen. Dieses Prinzip der Speicherorganisation – alle Teile eines DF in einem physikalisch zusammenhängenden Speicherbereich abzulegen – musste bei der Entwicklung der neuen Betriebssysteme aufgegeben werden, da die Speicherverwaltung sonst zu unflexibel geworden wäre.

Mechanismen der Freispeicherverwaltung

Der geringe Speicherplatz zwingt die Chipkarten-Betriebssysteme zu großen Einschränkungen in der Freispeicherverwaltung des EEPROMs. Erst seit Mitte der 90er-Jahre gibt es Betriebssysteme, die Dateien (EFs und DFs) nach der Personalisierung erzeugen und auch wieder löschen können. Dies muss aufgrund des Sicherheitscharakters einer Chipkarte natürlich immer in einer kryptografisch einwandfreien Art geschützt sein. Idealerweise werden die entsprechenden Kommandos nach einer einleitenden beidseitigen Authentisierung im Secure Messaging Modus ausgeführt.

[8] Siehe auch Abschnitt 3.4.3 „Zusatzhardware".

Eine Freispeicherverwaltung muss ebenfalls berücksichtigen, dass sich bei einem plötzlichen Stromausfall, wie er beispielsweise durch Ziehen der Karte zustande kommt, der gesamte Dateibaum nach wie vor in einem definierten Zustand befinden muss. Gerade in so einem Fall könnte die Sicherheit einer Chipkarte komplett zusammenbrechen, wenn Dateizeiger plötzlich undefiniert wären. Der adäquate Lösungsansatz sind hier wiederum atomare Abläufe, welche jedoch in diesem Fall aufgrund der großen Datenmengen verhältnismäßig viel Zeit und auch einen entsprechend großen Pufferspeicher benötigen.

Bei der Dateiverwaltung und im Übrigen auch bei Speicherverwaltungen aller Art mit Chipkarten gibt es unterschiedliche Realisierungsstrategien, die erhebliche Differenzen bei der softwaretechnischen Umsetzung haben. Anhand der Dateiverwaltung einer Chipkarte wird dies im Folgenden dargestellt.

Die einfachste Form der Speicherverwaltung ist eine Art WORM (*write once read multiple*)-Funktionalität. Der Speicherplatz einmal angelegter Dateien wird von diesen auch im (logisch) gelöschten Zustand weiterhin belegt. Der dazu notwendige Verwaltungsauf-

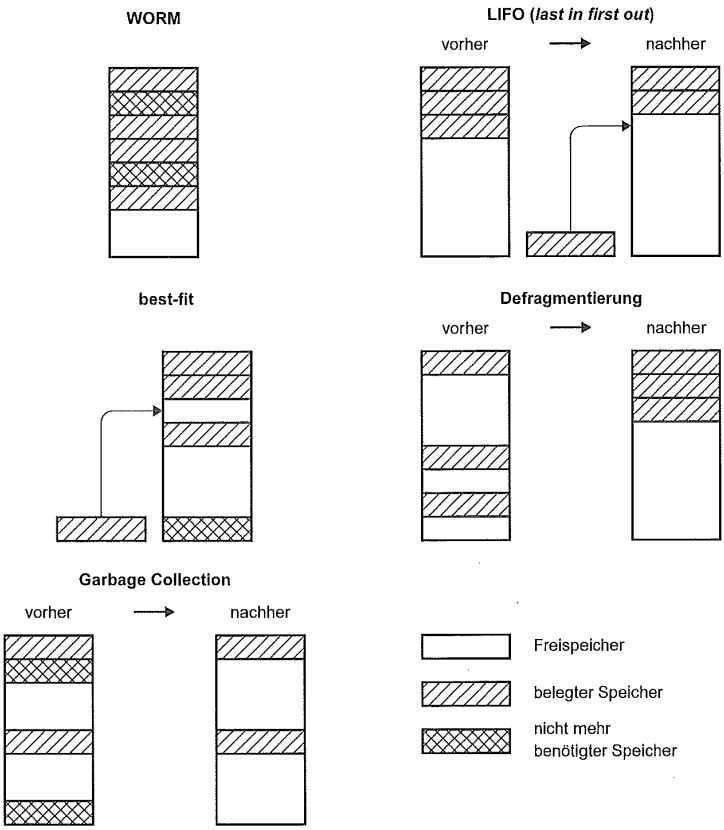


Bild 5.30 Darstellung der typischen Mechanismen für Speicherverwaltungen bei Chipkarten-Betriebssystemen. Eine detaillierte Erklärung findet sich im Text.

wand ist minimal. Informationstechnisch etwas aufwendiger verhält es sich mit einer Speicherverwaltung nach dem FIFO (*first in first out*)-Prinzip. Dabei kann jeweils die zuletzt erzeugte Datei unter Freigabe des vormals belegten Speicherplatzes wieder gelöscht werden. Dieses Verfahren findet oft bei einfachen Chipkarten-Betriebssystemen Anwendung. Bei dem etwas höher entwickelten Best-fit-Algorithmus sucht das Betriebssystem bei Anlage einer Datei immer den kleinsten dafür geeigneten Freispeicherbereich. Beim Löschen der Datei wird dieser Bereich wieder freigegeben und kann von einer anderen Datei benutzt werden. Werden jedoch Dateien verschiedener Größe oft erzeugt und wieder gelöscht, kommt es verhältnismäßig schnell zu einer starken Fragmentierung des Speichers. Das Ergebnis: Größere Dateien können nicht mehr angelegt werden, da kein zusammenhängender Speicher in der geforderten Größe mehr zur Verfügung steht, obwohl insgesamt noch genügend Speicher Freispeicher vorhanden ist. Genau hier setzt das Defragmentierungsverfahren an. Dieses für Chipkarten-Betriebssysteme verhältnismäßig aufwendige Speicherverwaltungsverfahren verschiebt im vorangehend beschriebenen Fall die vorhandenen Dateien so lange, bis der gesamte vorhandene Freispeicher zu einem zusammenhängenden Block verbunden ist. Die Schwierigkeit dabei ist die Tatsache, dass dieser Algorithmus aufgrund der vielen zeitintensiven EEPROM-Schreibzugriffe verhältnismäßig langsam abläuft.

Die Garbage Collection ist vom Prinzip unabhängig von den oben beschriebenen Verfahren zu sehen. Bei ihr wird bei Bedarf oder in festgelegten Zeitabständen der gesamte Speicher hinsichtlich nicht mehr benutzter Speicherblöcke durchsucht. Wird ein Block gefunden, der nicht mehr benötigt wird, ordnet ihn die Garbage Collection automatisch dem zur Verfügung stehenden Freispeicher zu. Über eine nachfolgende Defragmentierung kann man nun diese kleinen Speicherblöcke zu einem großen zusammenhängendem Freispeicherbereich zusammenschieben.

Datenintegrität

Ein weiterer wichtiger Punkt ist die Sicherstellung der Datenintegrität. Die Dateiverwaltung sollte zu jedem Zeitpunkt fähig sein zu prüfen, ob sich die Daten im Speicher unbeabsichtigt verändert haben, wie dies beispielsweise durch Alterung geschehen könnte. Um den Verwaltungsoverhead für diese Funktion zu minimieren, müssen nicht grundsätzlich alle Daten mit Prüfsummen gesichert sein, doch sollten mit der Wichtigkeit der Daten auch ihre Redundanz bzw. die überwachenden Schutzfunktionen wachsen. Abgesichert werden entweder mehrere Datenelemente auf einmal, wie beispielsweise ein gesamter Dateiheader, oder besonders wichtige Datenelemente jeweils separat. Dies hängt vor allem davon ab, wie häufig diese Datenelemente im EEPROM geändert werden und wie viel Speicherplatz man beim Design des Betriebssystems für die Integrität von Daten zu opfern bereit ist.

Um die Datenintegrität zu gewährleisten, werden vor allem kritische Datenelemente, wie Datenzugriffsrechte oder Zeiger vom Dateiheader zum Dateibody, mit Fehlererkennungscodes gesichert. Oft benutzt man dazu Prüfsummen auf der Grundlage von CRCs, da diese verhältnismäßig schnell zu berechnen sind und wenig Programmcode benötigen. Um jedoch die typische Ausfallcharakteristik von EEPROM-Zellen besser abzudecken, gebraucht man oft auch Reed-Solomon-Codes. Diese können die üblicherweise auftretenden Bündelfehler – d. h.: eine ganze EEPROM-Speicherseite hat sich verändert – wesentlich besser erkennen, als dies bei CRC-Prüfsummen der Fall ist.

Anwendungsübergreifende Zugriffe
Um bestimmte Funktionen auf einer Chipkarte freizuschalten, wird in der Regel eine vorherige PIN-Eingabe vom Kartenbenutzer verlangt. Da die Merkfähigkeit des Durchschnittsmenschen gewissen Beschränkungen unterliegt, ist es selbst bei mehreren Anwendungen auf einer Chipkarte üblich geworden, immer nur eine PIN pro Karte zu verwenden. Jede Anwendung auf der Karte benutzt diese gemeinsame PIN. Man könnte sie nun einmal pro Anwendung in einem internal EF speichern. Das Problem dabei wäre aber, dass dann jede dieser PINs einen eigenen Fehlbedienungszähler hätte. Bei beispielsweise fünf Anwendungen auf der Karte und jeweils drei erlaubten Fehlversuchen hätte man bereits 15 Versuche, um die PIN zu raten. Dies ist für Anwendungsdesign und -sicherheit in vielen Fällen nicht tolerabel. Deshalb bieten einige Betriebssysteme die Möglichkeit der anwendungsübergreifenden Zugriffe auf PINs und auch auf Schlüssel.

Diese Nutzung gemeinsamer Ressourcen wird dem Prinzip nach ähnlich realisiert wie die bei PC-Betriebssystemen üblichen Alias-Mechanismen. Der größte Unterschied liegt nur darin, dass bei Chipkarten-Betriebssystemen die Verweise immer nur zu den übergeordneten DFs hin möglich sind, mit dem MF als höchster Instanz. Es wäre also nicht möglich, auf eine PIN in einem anderen DF zuzugreifen, sondern nur auf eine PIN, die in der Datei-Hierarchie höher liegt, also beispielsweise im MF.

Bei obigem Beispiel mit einer von mehreren Anwendungen gemeinsam genutzten PIN inklusive Fehlbedienungszähler würde dies wie folgt realisiert: Die PIN wird in einem internal EF direkt unter dem MF abgelegt. In der Anwendung in einem DF unter dem MF befindet sich in einem internal EF ein Verweis auf den Speicherort der PIN. Die Ergebniszustände für erfolgreiche und fehlgeschlagene PIN-Vergleiche sind natürlich im PIN-Record im DF abgelegt, da sie jeweils nur eine bestimmte Anwendung betreffen. Wird nur ein PIN-Vergleich angestoßen, dann greift das Kommando VERIFY zuerst auf den PIN-Record im aktuellen DF zu, stellt dort fest, dass sich PIN und dazugehöriger Fehlbedienungszähler auf einer anderen Ebene befinden, und benutzt dann die so indizierte PIN für den Vergleich. Je nach Ergebnis des Vergleichs wird anschließend der im internal EF des aktuell selektierten DFs abgelegte Zustand für das DF gesetzt.

Dieses Verfahren wird mittlerweile von vielen Chipkarten-Betriebssystemen in unterschiedlichen Varianten unterstützt. Gerade für die Nutzung gemeinsamer Daten zwischen zwei oder mehreren Anwendungen stellt es eine sehr elegante und kryptografisch einwandfreie Lösung dar. Zusätzlich zu einer anwendungsübergreifenden Nutzung von PINs und Schlüsseln bieten manche Betriebssysteme auch einen äquivalenten Mechanismus für EFs an. Auf diese Weise ist es dann möglich, ohne die Deselektion eines DFs direkt auf globale Daten, die in EFs unter dem MFs gespeichert sind, zuzugreifen.

5.8 Ablaufsteuerung

Muss in einem Betriebssystem ein Zustandsautomat implementiert werden, kann dieser auf verschiedenartige Weise aufgebaut sein. Einige elementare Prinzipien sind jedoch unabhängig vom jeweiligen Betriebssystem und dessen Hersteller in jedem Falle einzuhalten.

Der Zustandsautomat muss sich im vorangehend beschriebenen Schichtenmodell des Betriebssystems nach dem Kommandointerpreter und vor der eigentlichen Ausführung des

Kommandos befinden. Seine Aufgabe ist es, anhand einer Tabelle festzustellen, ob das empfangene Kommando im aktuellen Zustand ausgeführt werden darf oder nicht. Grundprinzip dabei ist, wie im Chipkartenbereich üblich, ein möglichst geringer Verbrauch an Speicher für die Bereitstellung der Zustandsinformationen. Zusätzlich muss die Information aber so strukturiert sein, dass der Automat selber ebenfalls möglichst speichersparend aufgebaut werden kann.

Für die Analyse des im I/O-Puffer abgelegten Kommandos benötigt der Zustandsautomat einiges an Information. Im Bild 5.31 ist ein möglicher Aufbau einer Zustandstabelle für Chipkarten aufgeführt.

Das erste Datenelement (Initialzustand) beinhaltet den Zustand, für den der folgende Teil der Datenstruktur bearbeitet werden soll. Hier könnte eine Zahl stehen, die direkt den Zustand definiert, in dem alle weiteren Informationen berücksichtigt werden müssen. Dann folgen in einer Untertabelle die in dem Initialzustand erlaubten Kommandos. Dabei muss es möglich sein, jeweils einzelne Kommandos, Gruppen davon, keine oder alle Kommandos zuzulassen.

In der Tabellenstruktur folgen der Kommandodefinition die dazugehörigen erlaubten Parameter. In diesen Datenelementen muss auch die Möglichkeit vorhanden sein, sowohl Einzelwerte als auch Bereiche dafür zu definieren. Beispielsweise könnten im Feld für das Kommando die Codierung für READ BINARY stehen und in den Parameterfeldern 1 und 2 der minimale und maximale Offset für den Lesezugriff auf die transparente Datei. In Parameter 3 stünde dann sowohl die Grenze für die minimale als auch die maximale Länge. Da in dieser Untertabelle für einen Zustand auch mehrere Einträge vorhanden sein können, ist es möglich, dass nach READ BINARY noch weitere Kommandos mit allen ihren Parametern definiert sind.

Den Abschluss eines Eintrags in der Tabelle bildet der neue Zustand, der im Gutfall der Kommandoabarbeitung gesetzt wird, d. h. wenn das Kommando problemlos ausgeführt werden konnte. Im Schlechtfall lässt sich bei dieser Form der Datenstruktur ein anderer Zustand setzen. Um innerhalb des Zustandsautomaten möglichst flexibel zu bleiben, müssen dort sowohl absolute als auch relative positive und negative Folgezustände erlaubt sein. „Relativ" bedeutet in diesem Zusammenhang, dass der neue Zustand zustande kommt, indem auf den Initialzustand ein Wert addiert oder subtrahiert wird. Bei der absoluten Angabe wird der neue Wert direkt ohne Miteinbeziehung des Initialwertes gesetzt.

...	...	...	...
Initialzustand	Kommandodefinitionen	neuer Zustand im Gutfall	neuer Zustand im Schlechtfall
...	...	...	...

Kommandodefinitionen			
Kommando	Parameter P1 (min, max)	Parameter P2 (min, max)	Parameter P3 (min, max)
...	...	...	...
...	...	...	...

Bild 5.31 Beispiel für die Datenelemente einer Datenstruktur, die für einen Zustandsautomaten benötigt werden.

Die Möglichkeiten eines Zustandsautomaten sind prinzipiell unbegrenzt. Die hier aufgeführte Datenstruktur ist für den Einsatz in einem etwas höher entwickeltem Chipkarten-Betriebssystem ziemlich gut geeignet.

Mit der beschriebenen Datenstruktur und einem entsprechenden Zustandsautomaten kann prinzipiell jeder mögliche Zustandsgraph in einer Chipkarte nachgebildet werden. Natürlich schützen sich die einzelnen Dateien noch zusätzlich durch die Zugriffsbedingungen für die Kommandos gegen unbefugtes Lesen und Schreiben. Doch ergänzend zu dem objektorientierten Schutz der Dateien kann durch die Ablaufsteuerung der Kommandos ein weiterer übergeordneter Mechanismus zur Erhöhung der Systemsicherheit bereitgestellt werden. Dies ist der eigentliche große Vorteil von Zustandsautomaten in Chipkarten.

5.9 Ressourcenzugriffe nach ISO/IEC 7816-9

Bei der Festlegung der erlaubten Zugriffe auf Dateien kann zwischen zustands- und kommandoorientierten Zugriffsbedingungen unterschieden werden. Bei den zustandsorientierten Zugriffsbedingungen wird der aktuelle Sicherheitszustand verknüpft mit einem definierbaren logischen Vergleich mit der entsprechenden Zugriffsbedingung der Datei verglichen. Bei den aktuellen Sicherheitszuständen gibt es die Variante globaler Sicherheitszustand (d. h. die Chipkarte als Ganzes) und lokaler Sicherheitszustand (d. h. aktuell selektiertes Verzeichnis). Bei kommandoorientierten Zugriffsbedingungen hingegen enthält die Zugriffstabelle in der Datei Informationen über die zuvor erfolgreich auszuführenden Kommandos für jede der Zugriffsarten.

Sowohl zustands- als auch kommandoorientierte Zugriffsbedingungen wurden und werden in unterschiedlicher Ausprägung von den am Markt befindlichen Chipkarten-Betriebssystemen unterstützt. Das große Manko waren bislang die Mannigfaltigkeit der Realisierungen und Lösungsansätze. Die ISO/IEC 7816-9 hat die Aufgabe, Zugriffe auf Ressourcen einer Chipkarte zu vereinheitlichen, und besitzt eigens zu diesem Thema einen Abschnitt, der ein sehr mächtiges, doch leider auch kompliziertes Modell für Zugriffsbedingungen

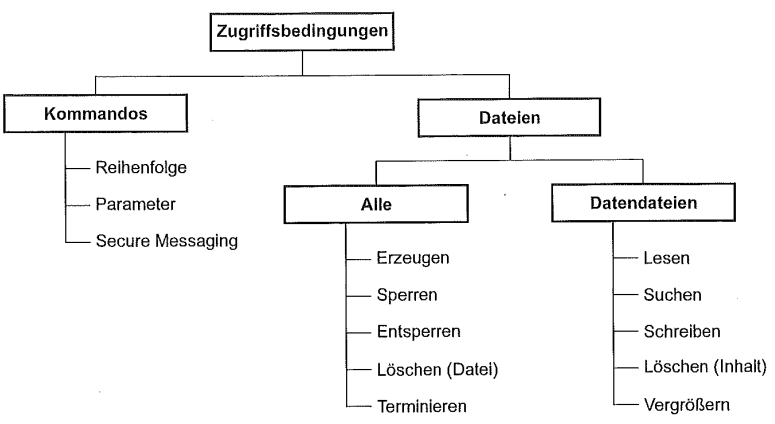


Bild 5.32 Klassifizierungsbaum der Zugriffsbedingungen für Kommandos und Dateien gemäß ISO/IEC 7816-9.

sowohl auf Dateien als auch Kommandos und Datenobjekte festlegt. Dieses universelle Zugriffsmodell vereint sowohl zustands- als auch kommandoorientierte Zugriffsbedingungen und fügt zusätzlich noch die Möglichkeiten der Festlegung von bestimmten Kommandosequenzen hinzu. Die ISO/IEC 7819-9 lässt des Weiteren auch die Möglichkeit offen, über bestimmte Kennzeichen bei den Datenobjekten einen Zustandsautomaten für die Zugriffe festzulegen. Das Konzept baut vollständig auf TLV-codierten Datenobjekten auf, die sich bekanntermaßen sehr flexibel und informationstechnisch elegant einsetzen lassen.

Die ISO/IEC 7819-9 definiert so genannte Sicherheitsattribute (*security attributes* – SA), die zur Regelung des Zugriffs, Nicht-Zugriffs und zur Erreichung von bestimmten Sicherheitszuständen in der Chipkarte verwendet werden können. Diese Sicherheitsattribute steuern den Zugriff auf Kartenressourcen, wie Dateien, Kommandos, Datenobjekte sowie bei SCQL-Tabellen und Views.

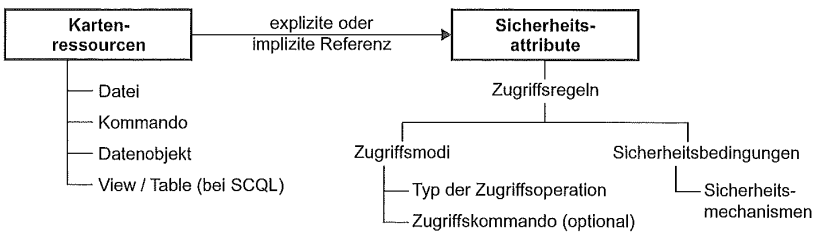


Bild 5.33 Klassifizierungsbäume der möglichen Kartenressourcen und dazugehörigen Sicherheitsattribute nach ISO/IEC 7816-9.

Das Prinzip der Zugriffssteuerung ist dabei relativ einfach. Die zu schützende Ressource erhält entweder einen impliziten oder einen expliziten Verweis auf ein oder mehrere Sicherheitsattribute. Diese bestehen aus einer oder mehreren Zugriffsregeln (*access rules* – AR), die sich ihrerseits wiederum aus Zugriffsmode (*access mode* – AM) und Sicherheitsbedingungen (*security conditions* – SC) zusammensetzen. Der Zugriffsmode legt die Art des Zugriffs – also beispielsweise lesend oder schreibend – fest und die Sicherheitsbedingungen den notwendigen Sicherheitsmechanismus (*security mechanism* – SM), um die Zugriffsbedingungen zu erreichen. Als zusätzliches Element ist es noch möglich, die jeweilige Sicherheitsumgebung (*security environment* – SE) in die Zugriffsregeln mit einfließen zu lassen.

Alle ISO/IEC 7816-9 Sicherheitsattribute sind entweder speicherplatzsparend in einem Compact-Format oder in regulär TLV-codierten Datenobjekten im Expanded-Format gespeichert. Beide Formate stellen eine ähnliche Funktionalität des Zugriffsschutzes zu Ver-

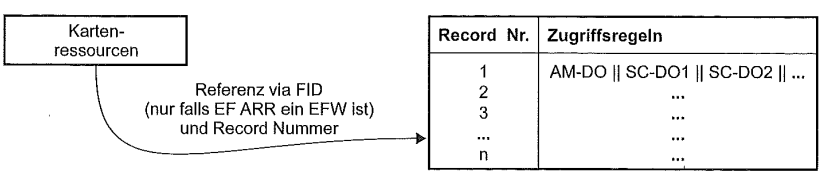


Bild 5.34 Darstellung der Verbindung von beliebigen Ressourcen einer Chipkarte mit den dazugehörigen Zugriffsregeln, gespeichert in einer Datei EF_{ARR}.

fügung, wobei jedoch das Expanded-Format noch deutlich mehr Variationsmöglichkeiten bietet. So kann mit diesem Format beispielsweise auch eine detaillierte Festlegung von Kommandos und deren Parametern zum Zugriff durchgeführt werden.

Abgelegt werden die Zugriffsregeln in einem oder mehreren EFs mit der Struktur linear variable. Dieses kann ein EFI (EF internal) oder ein EFW (EF working) sein und hat den Namen EF_{ARR} (*access rule reference*). Sein FID (*file identifier*) kann frei gewählt werden. Wird ein EFI zur Speicherung der Zugriffsregeln benutzt, so wird implizit bei jeder auftretenden Zugriffsbedingung über das Chipkarten-Betriebssystem darauf zugegriffen. Bei einem EFW existiert in der zu schützenden Kartenressource eine Referenz auf den FID des EF_{ARR}. In beiden Fällen referiert die zu schützende Karten-Ressource auf die Nummer des Records im EF_{ARR}, der die entsprechende Zugriffsregel enthält. Der Vorteil eines selektierbaren – d. h. EFW – EF_{ARR} liegt vor allem darin, dass dessen Inhalt auch mit den üblichen Kommandos und entsprechenden Zugriffsbedingungen verändert werden kann. Dies schafft eine enorme Flexibilität, da so zu beliebigen Zeitpunkten die Zugriffsbedingungen auf die Ressourcen der Chipkarte angepasst werden können.

Wichtig ist zu erwähnen, dass natürlich nicht für jedes EF ein eigener Record im EF_{ARR} angelegt werden muss. Es reicht völlig aus, einen einzigen Record für alle EF mit identischen Zugriffsbedingungen anzulegen und dann von diesen EFs darauf zu referieren. Dies reduziert die Anzahl der notwendigen Records im EF_{ARR} erheblich.

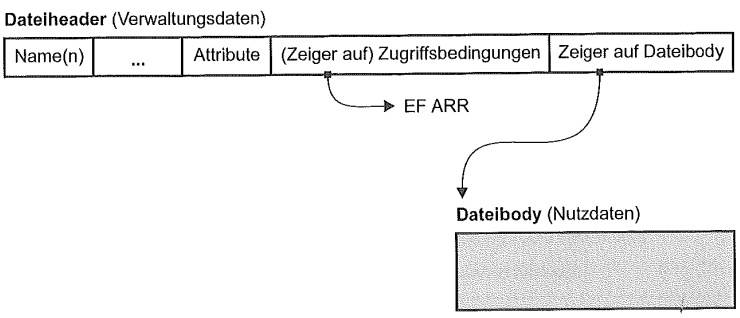


Bild 5.35 Darstellung der Verbindung eines beliebigen Dateiheaders in einem Dateiverwaltungssystem einer Chipkarte mit den dazugehörigen Zugriffsregeln, gespeichert in einer Datei EF_{ARR}.

Die Verbindung zwischen EF zu EF_{ARR} besteht nur in einer Richtung und ist nicht etwa zweiseitig. Es ist deshalb nicht möglich, von einem Record im EF_{ARR} auf das oder die darauf referierenden EFs zu schließen. Dies ist bei Dateiverwaltungen von Belang, denn wenn ein EF gelöscht wird, ist es nicht erlaubt, den dazugehörigen Record im EF_{ARR} zu löschen, da es möglich ist, dass noch eines oder mehrere andere EFs auf die betreffende Regel referieren.

Beim Zugriff auf eine Datei mit einem Kommando wird nun der folgende Ablauf durchgeführt: Zuerst prüft das Betriebssystem, ob das explizit oder implizit referierte EF_{ARR} und der entsprechende Record vorhanden sind. Ist dies nicht der Fall, so wird der Zugriff verweigert. Anschließend wird im EF_{ARR} nach einem AM (*access mode*) – Datenobjekt für den geforderten Zugriff gesucht. Falls dieses gefunden wurde, wird die angegebene

Sicherheitsbedingung (SC – *security condition*) geprüft, andernfalls der Zugriff wiederum verweigert. Ist die geforderte Sicherheitsbedingung erfüllt, darf auf die Datei mit dem entsprechenden Kommando zugegriffen werden.

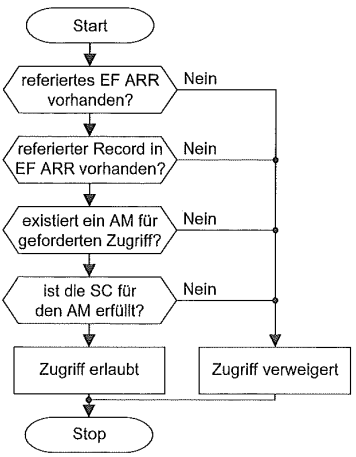


Bild 5.36 Flussdiagramm mit den wesentlichen Abfragen bei der Prüfung auf Dateizugriff gemäß dem Zugriffsmodell nach ISO/IEC 7816-9.

Tabelle 5.10 Aufstellung der möglichen Codierungen für das AM (*access mode*)-Byte für DFs nach ISO/IEC 7816-9

b8	b7	b6	b5	b4	b3	b2	b1	Bedeutung
0	...	...	...	...	...	...	...	b7 . . . b1 gemäß dieser Tabelle
1	...	...	...	...	...	...	...	b3 . . . b1 gemäß dieser Tabelle und b7 . . . b4 spezifisch (*proprietary*)
...	1	...	...	...	...	...	...	DELETE FILE (diese Datei)
...	...	1	...	...	...	...	...	TERMINATE CARD USAGE (MF), TERMINATE DF
...	...	...	1	...	...	...	...	ACTIVATE FILE
...	...	...	...	1	...	...	...	DEACTIVATE FILE
...	...	...	...	...	1	...	...	CREATE FILE (Erzeugung eines DFs)
...	...	...	...	...	...	1	...	CREATE FILE (Erzeugung eines EFs)
...	...	...	...	...	...	...	1	DELETE FILE (untergeordnete Datei)

Tabelle 5.11 Aufstellung der möglichen Codierungen für das AM (*access mode*)-Byte für EFs nach ISO/IEC 7816-9

b8	b7	b6	b5	b4	b3	b2	b1	Bedeutung
0	...	...	...	...	...	...	...	b7 . . . b1 gemäß dieser Tabelle
1	...	...	...	...	...	...	...	b3 . . . b1 gemäß dieser Tabelle und b7 . . . b4 spezifisch (*proprietary*)
...	1	...	...	...	...	...	...	DELETE FILE
...	...	1	...	...	...	...	...	TERMINATE EF
...	...	...	1	...	...	...	...	ACTIVATE FILE
...	...	...	...	1	...	...	...	DEACTIVATE FILE
...	...	...	...	...	1	...	...	WRITE BINARY, WRITE RECORD, APPEND RECORD
...	...	...	...	...	...	1	...	UPDATE BINARY, UPDATE RECORD, ERASE BINARY
...	...	...	...	...	...	...	1	READ BINARY, READ RECORD, SEARCH BINARY, SEARCH RECORD

Tabelle 5.12 Aufstellung der möglichen Codierungen für die Sicherheitsbedingungen (SC – *security condition*) nach ISO/IEC 7816-9

b8	b7	b6	b5	b4	b3	b2	b1	Bedeutung
0	0	0	0	0	0	0	0	Ein Zugriff ist immer erlaubt (*always*).
1	1	1	1	1	1	1	1	Ein Zugriff ist nie erlaubt (*never*).
...	...	...	...	0000				keine Referenz auf Security Environment
...	...	...	...	0001 ... 1110				Nummer des Security Environments
...	...	...	...	1111				RFU
0	...	...	...	...	...	...	...	mindestens eine Bedingung muss erfüllt sein
1	...	...	...	...	...	...	...	alle Bedingungen müssen erfüllt sein
...	1	...	...	...	...	...	...	Secure Messaging
...	...	1	...	...	...	...	...	EXTERNAL AUTHENTICATION
...	...	...	1	...	...	...	...	Benutzerauthentisierung (z. B. PIN-Eingabe)

Tabelle 5.13 Aufstellung der möglichen Codierungen für die Datenobjekte des Zugriffsmodus (AM DO – *access mode data object*) nach ISO/IEC 7816-9. Diese werden im Extended-Format verwendet.

Kennzeichen	Länge	Bedeutung
'80'	1	AM-Byte aus den Tabellen 5.10 und 5.11.
'81' ... '8F'	x	Beschreibung von CLA || INS || P1 || P2 zur Festlegung der Parameter von Zugriffskommandos
'9C'	x	Spezifische (*proprietary*) Beschreibung eines Zustandsautomaten

Tabelle 5.14 Aufstellung der möglichen Codierungen für die Datenobjekte der Sicherheitsbedingungen (SC DO – *security condition data object*) nach ISO/IEC 7816-9. Diese werden im Extended-Format verwendet.

Kennzeichen	Länge	Bedeutung
'90'	0	Ein Zugriff ist immer erlaubt (*always*).
'97'	0	Ein Zugriff ist nie erlaubt (*never*).
'A4'	x	CRT (*control reference template)* für Authentisierung (EXTERNAL AUTHENTICATION oder Benutzerauthentisierung)
'B4', 'B6', 'B8'	x	CRT (*control reference template)* für Kommando und/oder Antwort mit Secure Messaging
'9E'	x	Sicherheitsbedingung nach Tabelle 5.15.
'A0'	x	Die aufgeführten Sicherheitsbedingungen sind mit einem logischen ODER miteinander zu verknüpfen.
'AF'	x	Die aufgeführten Sicherheitsbedingungen sind mit einem logischen UND miteinander zu verknüpfen.

Tabelle 5.15 Aufstellung der möglichen Codierungen für die Datenobjekte zur Nutzungscharakterisierung (*usage qualifier*) im Rahmen eines CRT (*control reference template*) nach ISO/IEC 7816-9. Diese werden im Extended-Format verwendet. Das Kennzeichen für Nutzungscharakterisierung ist '95'.

b8	b7	b6	b5	b4	b3	b2	b1	Bedeutung
1	...	...	...	...	...	...	...	Überprüfung, Verschlüsselung, EXTERNAL AUTHENTICATION
...	1	...	...	...	...	...	...	Berechnung (*computation*), Entschlüsselung, INTERNAL AUTHENTICATION
...	...	1	...	...	...	...	...	Secure Messaging Antwort (*response*)
...	...	...	1	...	...	...	...	Secure Messaging Kommando (*command*)
...	...	...	...	1	...	...	...	wissensbasierte Benutzerauthentisierung (z. B. PIN)
...	...	...	...	...	1	...	...	biometrische Benutzerauthentisierung
...	...	...	...	...	...	x	x	RFU

Tabelle 5.16 Aufstellung von möglichen Codierungen für die Datenverwaltungsparameter (FCP – *file control parameter*) nach ISO/IEC 7816-4 und ISO/IEC 7816-9. Das Kennzeichen für FCP ist '62'.

Kennzeichen	Länge	Bedeutung
'80'	2	Anzahl der Datenbytes ohne Strukturinformationen in transparenten EFs
'81'	2	Anzahl der Datenbytes mit Strukturinformationen.
'83'	2	FID
'84'	1 . . . 16	DF Name
'88'	1	SFI codiert in den Bits 8 bis 4, Bits 3 . . . 1 sind auf 0 gesetzt.
'8A'	1	Statuswert des Lebenszyklus (LCSI – *life cycle status integer*)
'8C'	variabel	Sicherheitsattribute im compact format
'AB'	variabel	Sicherheitsattribute im expanded format

Tabelle 5.17 Aufstellung von möglichen Codierungen für Control Reference Datenobjekte (CR DO – *control reference data object*) nach ISO/IEC 7816-4.

Kennzeichen	Bedeutung
'80'	Verweis auf Algorithmus (*algorithm reference*)
'81'	Verweis auf Dateien: FID oder Pfad zur Datei
'82'	Verweis auf Dateien: DF Name
'83'	Verweis auf Schlüssel: zum direkten Gebrauch (*for direct use*)
'84'	Verweis auf Schlüssel: zur Berechnung eines Sitzungsschlüssels

Zur Verdeutlichung der obigen Erklärungen seien in den folgenden Tabellen einige typische Beispiele für Einträge in einem EF_{ARR} mit vollständiger Codierung sowohl im Compact Format als auch im Expanded Format angegeben. Zusammenfassend ist zu den Zugriffsregeln nach ISO/IEC 7816-9 zu bemerken, dass dieses System zwar sehr mächtig und hochflexibel ist, doch auch einen entsprechenden Tribut an zusätzlichen Speicher vom Chipkarten-Betriebs-

Tabelle 5.18 Beispiel für einen Recordinhalt im Compact Format in einem EF_{ARR}, in dem die Zugriffsbedingungen für UPDATE BINARY, . . . und READ BINARY, . . . für eine Datei (EF) festgelegt sind. Alle anderen Dateizugriffe sind dann automatisch verboten.

Datum	Bezeichnung	Bedeutung
'8C'	Kennzeichen	Das Kennzeichen '8C' leitet eine Zugriffsregel im Compact Format ein.
'03'	Länge	Die Länge der folgenden Daten ist 3 Byte.
'03'	AM	Die folgenden Sicherheitsbedingungen beziehen sich auf UPDATE BINARY, . . . und READ BINARY, . . ., da '03' = °0000 0011°.
'00'	SC	Eine Sicherheitsbedingung für UPDATE BINARY, . . . ist nicht gegeben, das heißt, das EF darf immer geschrieben werden.
'00'	SC	Eine Sicherheitsbedingung für READ BINARY, . . . ist nicht gegeben, das heißt, das EF darf immer gelesen werden.

Tabelle 5.19 Beispiel für einen Recordinhalt im Compact Format in einem EF_{ARR}, in dem die Zugriffsbedingungen für ACTIVATE FILE, DEACTIVATE FILE und READ BINARY für eine Datei (EF) festgelegt sind. Alle anderen Dateizugriffe sind dann automatisch verboten.

Datum	Bezeichnung	Bedeutung
'8C'	Kennzeichen	Das Kennzeichen '8C' leitet eine Zugriffsregel im Compact Format ein.
'04'	Länge	Die Länge der folgenden Daten ist 4 Byte.
'19'	AM	Die folgenden Sicherheitsbedingungen beziehen sich auf ACTIVATE FILE, DEACTIVATE FILE und READ BINARY, . . ., da '19' = °0001 1001°.
'90'	SC	Als Sicherheitsbedingung für ACTIVATE FILE ist eine vorherige Benutzerauthentisierung notwendig, z. B. PIN Eingabe, da '90' = °1001 0000°. Die dazu notwendige PIN ist dem Betriebssystem implizit bekannt.
'90'	SC	Als Sicherheitsbedingung für DEACTIVATE FILE ist eine vorherige Benutzerauthentisierung notwendig, z. B. PIN Eingabe, da '90' = °1001 0000°. Die dazu notwendige PIN ist dem Betriebssystem implizit bekannt.
'00'	SC	Eine Sicherheitsbedingung für READ BINARY, . . . ist nicht gegeben, das heißt, das EF darf immer gelesen werden.

system fordert. Zusätzlich ist es mit diesen regelbasierten Zugriffsbedingungen in der Praxis beinahe unmöglich geworden, die Zugriffsbedingungen selbst für eine einfache Chipkarten-Anwendung ohne Unterstützung von entsprechenden Softwarewerkzeugen manuell zu codieren. Trotzdem wird sich dieses Konzept für Zugriffe auf Ressourcen einer Chipkarte weltweit durchsetzen, da die Vorteile der Flexibilität und Vereinheitlichung schwerer wiegen.

Tabelle 5.20 Beispiel für den Inhalt eines Records im Expanded Format in einem EFARR nach ISO/IEC 7816-9. Die folgende Festlegung erlaubt Zugriffe mit READ BINARY, ... auf das entsprechende EF immer. UPDATE BINARY, . . . ist nur nach vorheriger erfolgreicher Überprüfung von PIN 1 oder PIN 2 erlaubt. Alle anderen Dateizugriffe sind damit automatisch verboten.

Datum	Bezeichnung	Bedeutung
'AB'	Kennzeichen	Das Kennzeichen 'AB' leitet eine Zugriffsregel im Expanded Format ein.
'1A'	Länge	Die Länge der folgenden Daten ist 26 Byte.
'80'	AM DO	Das Kennzeichen '80' zeigt an, dass das Datenobjekt für die Zugriffsbedingungen AM DO ein Byte mit den Zugriffsbedingungen (AM Byte) enthält.
'01'	Länge	Die Länge der folgenden Daten ist 1 Byte.
'02'	AM	Die folgenden Sicherheitsbedingungen beziehen sich auf UPDATE BINARY, ..., da '02' = °0000 0010°.
'A0'	SC	Als Sicherheitsbedingung für die folgenden Sicherheitsbedingungen SC sind miteinander ODER verknüpft.
'10'	Länge	Die Länge der folgenden Daten ist 16 Byte (= '10').
'A4'	CRT	Das Kennzeichen 'A4' zeigt an, dass die folgenden Daten Informationen über notwendige Authentisierungen enthalten. Diese Daten sind ein CRT (*control reference template*).
'06'	Länge	Die Länge der folgenden Daten ist 6 Byte.
'83'	Key Reference	Das Kennzeichen '83' zeigt an, dass es sich um ein CRT-Datenobjekt zur Schlüsselreferenzierung handelt.
'01'	Länge	Die Länge der folgenden Daten ist 1 Byte.
'01'	Schlüsselnummer	Es ist der Schlüssel mit der Nummer 1 zu verwenden.
'95'	Usage Qualifier DO	Das Kennzeichen '95' = °1001 0000° zeigt an, dass es sich um ein CRT-Datenobjekt zur Nutzungscharakterisierung (*usage qualifier*) handelt.
'01'	Länge	Die Länge der folgenden Daten ist 1 Byte.
'08'	Usage Qualifier	Als Nutzungscharakterisierung ist eine wissensbasierte Benutzerauthentisierung (d. h. PIN) festgelegt, da '08' = °0000 1000°.
'A4'	CRT DO	Das Kennzeichen 'A4' zeigt an, dass die folgenden Daten Informationen über notwendige Authentisierungen enthalten. Diese Daten sind ein CRT (*control reference template*).
'06'	Länge	Die Länge der folgenden Daten ist 6 Byte.
'83'	Key Reference	Das Kennzeichen '83' zeigt an, dass es sich um ein CRT-Datenobjekt zur Schlüsselreferenzierung handelt.
'01'	Länge	Die Länge der folgenden Daten ist 1 Byte.
'02'	Schlüsselnummer	Es ist der Schlüssel mit der Nummer 2 zu verwenden.

Tabelle 5.20 *(Fortsetzung)*

Datum	Bezeichnung	Bedeutung
'95'	Usage Qualifier DO	Das Kennzeichen '95' = °1001 0000° zeigt an, dass es sich um ein CRT-Datenobjekt zur Nutzungscharakterisierung (*usage qualifier*) handelt.
'01'	Länge	Die Länge der folgenden Daten ist 1 Byte.
'08'	Usage Qualifier	Als Nutzungscharakterisierung ist eine wissensbasierte Benutzerauthentisierung (d. h. PIN) festgelegt, da '08' = °0000 1000°.
'80'	AM DO	Das Kennzeichen '80' zeigt an, dass es sich um ein AM-Datenobjekt (*access mode*) handelt.
'01'	Länge	Die Länge der folgenden Daten ist 1 Byte.
'01'	AM	Die Sicherheitsbedingungen beziehen sich auf READ BINARY, . . ., da '01' = °0000 0001°.
'90'	SC DO	Das Kennzeichen '90' zeigt an, dass Zugriffe immer erlaubt sind (*always*).
'00'	Länge	Die Länge der folgenden Daten ist 0 Byte. Daraus folgt, dass eine Sicherheitsbedingung für READ BINARY, . . . nicht notwendig ist, das heißt, das EF darf immer gelesen werden.

5.10 Atomare Abläufe

Oftmals wird für die Software im Mikrocontroller der Chipkarte gefordert, dass bestimmte Teile von ihr entweder vollständig oder gar nicht durchlaufen werden. Abläufe, die nicht aufteilbar sind und diese Forderung erfüllen, nennt man deshalb auch atomare Abläufe. Sie treten immer in Verbindung mit EEPROM-Schreibroutinen auf.

Atomaren Abläufen liegt der Gedanke zugrunde, dass man beim Schreiben ins EEPROM sicherstellen will, dass die betreffenden Daten in keinem Fall nur teilweise geschrieben worden sind. Dies würde beispielsweise dann der Fall sein, wenn der Benutzer die Chipkarte im falschen Augenblick aus dem Terminal zieht oder es zu einem plötzlichen Stromausfall kommt. Da die Chipkarte über keinerlei Pufferspeicher für elektrische Energie verfügt, verliert die Software in der Karte in diesen Fällen sofort ihre Aktionsmöglichkeiten.

Vor allem bei elektronischen Geldbörsen auf Chipkarten muss unter allen Umständen sichergestellt sein, dass die Einträge in den Dateien vollständig und korrekt sind. Beispielsweise wäre es äußerst fatal, wenn der Saldo einer Börse durch das Ziehen der Karte aus dem Terminal nur unvollständig auf den neuesten Stand gebracht würde. Auch müssen die entsprechenden Einträge in den Protokolldateien immer vollständig sein. Da die Hardware von Chipkarten atomare Abläufe nicht unterstützt, müssen diese deshalb mit Software realisiert werden. Die dazu angewandten Methoden sind im Prinzip nicht neu, sondern werden im Bereich von Datenbanken und Festplattenlaufwerken schon seit langer Zeit eingesetzt. Eine Variante, die bei Chipkarten-Betriebssystemen Verwendung findet, ist nachfolgend in ihren grundlegenden Abläufen beschrieben. Dieses Fehlerbehebungsverfahren (*error recovery*) ist transparent zur äußeren Welt und erfordert dadurch keinerlei Änderungen in eventuell bestehenden Anwendungen.

Man nehme zur Demonstration der Methode an, dass über die Schnittstelle zur Karte Daten gesandt werden, welche für eine Datei bestimmt sind. Dies wäre beispielsweise der typische Ablauf bei einem UPDATE BINARY-Kommando. Anhand von Bild 5.37 und der folgenden Erklärung lassen sich nun die konkreten Abläufe in der Karte nachverfolgen. Im EEPROM-Teil des Betriebssystems ist ein Puffer eingerichtet, der ausreichend groß ist, um alle notwendigen Daten aufzunehmen. Dieser besitzt, ebenfalls im EEPROM, ein Kennzeichen für den Zustand. Dieses kann entweder auf „Daten im Puffer gültig" oder auf „Daten im Puffer ungültig" gesetzt sein. Zusätzlich zum Puffer muss noch ein entsprechender Speicher für die Zieladresse und die aktuelle Länge der Pufferdaten vorhanden sein.

Der konkrete Ablauf stellt sich nun folgendermaßen dar: Die Daten, die sich ab der Zieladresse befinden, beispielsweise in einer Datei, werden im ersten Schritt unter Angabe der physikalischen Adresse und ihrer Länge in den Puffer kopiert. Nun wird das Flag für den Pufferinhalt auf „Daten im Puffer gültig" gesetzt. Im nächsten Schritt kopiert das Betriebssystem die neuen Daten an die gewünschte Adresse und ändert das Flag wieder auf „Daten im Puffer ungültig". Beim Hochfahren des Betriebssystems vor dem ATR wird das Flag abgefragt. Ist es auf „Daten im Puffer gültig" gesetzt, findet automatisch ein Schreibvorgang der im Puffer befindlichen Daten in den ebenfalls gespeicherten Adressbereich statt.

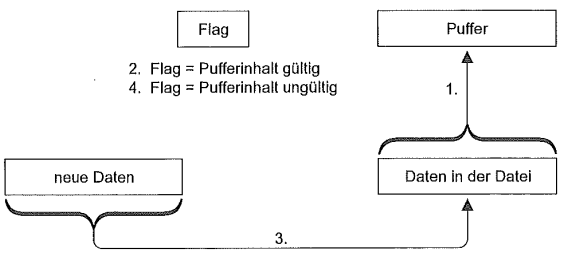


Bild 5.37 Beispiel einer möglichen Realisierung von atomaren Abläufen im Betriebssystem einer Chipkarte. Das Verfahren kann natürlich für die parallele Verarbeitung mehrerer Datenelemente kaskadiert werden.

Mit diesem Mechanismus ist somit in jedem Fall gewährleistet, dass gültige Daten in der Datei stehen. Kommt es an irgendeiner Stelle im Programmablauf zum Abbruch der Routine, so können die Daten im EEPROM der Chipkarte restauriert werden. Zieht beispielsweise in Schritt 3 – dem Schreiben der neuen Daten ins EEPROM – der Benutzer die Karte, so stehen die neuen Daten nur teilweise in der Datei. Beim Anschalten der Karte bei der nächsten Sitzung merkt das Betriebssystem, dass sich gültige Daten im Puffer befinden, und kopiert diese an die entsprechende Stelle. Damit ist der ursprüngliche Zustand wiederhergestellt, und alle Einträge in den Dateien im EEPROM sind konsistent. Als Zeitpunkt für diese Korrekturen eignet sich sehr gut die initial waiting time zwischen den einzelnen Bytes im ATR.[9]

[9] Siehe auch Abschnitt 6.2 „Answer to Reset – ATR".

Der beschriebene Ablauf hat jedoch zwei gravierende Nachteile: Der Pufferspeicher wird am stärksten von allen EEPROM-Bereichen durch Schreiben und Löschen belastet. Weil die Anzahl der Schreib-/Löschzyklen des EEPROMs begrenzt ist, erhält man in dem wichtigen Puffer mit hoher Wahrscheinlichkeit als ersten EEPROM-Bereich einen Schreibfehler. Dies würde bedeuten, dass man die Chipkarte nicht mehr verwenden kann, da die Datenkonsistenz nicht mehr sichergestellt ist. Dieses Problem kann entschärft werden, indem man den Puffer zyklisch aufbaut, sodass nicht immer an die gleiche Stelle geschrieben werden muss. Leider benötigt man dann für den Puffer unverhältnismäßig viel EEPROM. Ein weiterer Nachteil dieser Realisierung von atomaren Abläufen ist die Verlängerung der Programmausführungszeiten aufgrund der obligatorischen Schreibzugriffe in den Pufferspeicher. In ungünstigen Fällen dauert aufgrund dieses Mechanismus der Zugriff dreimal länger als bei direktem Schreiben in den EEPROM-Speicher. Deshalb ist es üblich, nicht alle EEPROM-Zugriffe zu puffern, sondern nur Schreibzugriffe auf bestimmte Dateien oder Datenelemente im EEPROM. Dies kann bei Dateien beispielsweise durch Attribute im Header jeweils festgelegt werden.

Obiges Verfahren kann sehr einfach dahingehend erweitert werden, dass jeweils nicht nur ein Datenelement in den Puffer geschrieben wird, sondern mehrere. Dann ist es sogar möglich, Schreibzugriffe auf mehrere unterschiedliche Dateien oder Datenelemente entweder ganz oder gar nicht auszuführen. Moderne Chipkarten-Betriebssysteme wie Java Card unterstützen beinahe ausnahmslos atomare Abläufe oder bieten auch die Möglichkeit, längere Sequenzen im Programmablauf als atomar zu kennzeichnen und damit vor Stromabbrüchen zu schützen.

5.11 Open Platform

Visa International war durch seine Aufgaben als einer der größten Kartenherausgeber schon frühzeitig mit der Problematik der Verwaltung mannigfacher Anwendungen unterschiedlichster Herausgeber auf Multiapplikations-Chipkarten konfrontiert. Dadurch kam es zur Erstellung der Visa Open Platform (VOP)-Spezifikation, welche eine Schnittstelle innerhalb von Chipkarten-Betriebssystemen mit der Aufgabe der Verwaltung von Chipkarten-Anwendungen ist. Der Herausgeber dieser Spezifikation ist seit 1999 das Global Plattform Gremium [Global Platform], das als Aufgabe die Standardisierung von Technologien für Multiapplication-Chipkarten hat. Zu diesem Zeitpunkt wurde auch der Name in Open Platform (OP) geändert. Die OP-Spezifikation ist der wichtigste internationale Standard für Applikationsmanagement bei Multiapplication-Chipkarten und kann vom Web-Server des Global Platform Gremiums frei bezogen werden.

Die OP-Spezifikation ist vom Ansatz her unabhängig von einem bestimmten Betriebssystem, sodass damit sowohl proprietäre Chipkarten-Betriebssysteme als auch offene Plattformen wie Multos oder Java Card unterstützt werden können. In der Praxis hat sich jedoch vor allem bei Java Card die OP-Spezifikation als *De facto*-Standard zur Verwaltung und zum Laden von Java-basierten Anwendungen etabliert. So sieht beispielsweise die ETSI Norm GSM 03.19 Open Platform als Standard zum Nachladen von Anwendungen vor.

Der Zweck der OP-Spezifikation besteht darin, einem Kartenherausgeber Mechanismen zur sicheren Verwaltung von Anwendungen Dritter auf der von ihm emittierten Chipkarte an die Hand zu geben. Dazu definiert Open Platform die grundlegende Architektur einer Multiapplikations-Chipkarte. Diese ist in Bild 5.38 aufgezeigt und für das Verständnis der Mechanismen von großer Bedeutung.

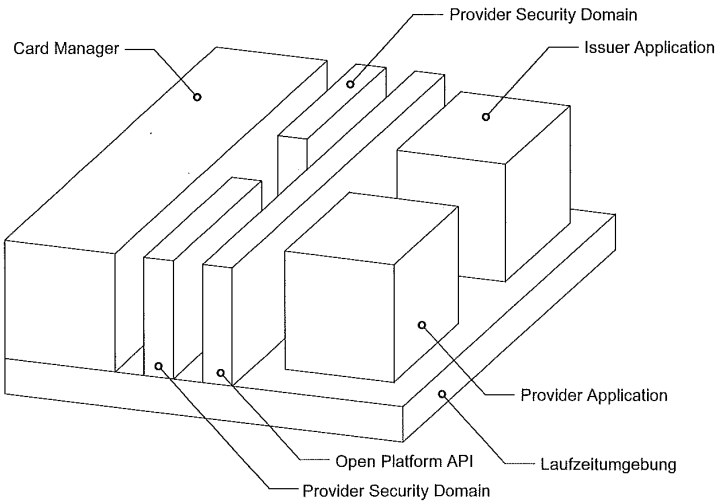


Bild 5.38 Grundlegende Architektur und Komponenten der Open Platform.

Die Basis für alle Anwendungen bildet die Laufzeitumgebung, welche eine Anwendungsseparierung, eine Hardware-unabhängige Schnittstelle (API) sowie Speicherplatz für die Daten und Programme der verschiedenen Anwendungen zur Verfügung stellt. Aufbauend auf diesem Fundament sitzt der Card Manager. Er ist die zentrale Komponente von Open Platform und über eine frei wählbare AID selektierbar. Der Card Manager ist sozusagen der informationstechnische Repräsentant des Kartenherausgebers auf der Multiapplikations-Chipkarte. Er verwaltet die Laufzeitumgebung hinsichtlich der Anwendungen und stellt diesen auch Schnittstellen für oncard-Dienste als auch zur äußeren Welt zur Verfügung. Dies schließt auch die von offcard kommende Auswahl von Anwendungen auf der Chipkarte mit ein und auch die Verteilung der von der äußeren Welt kommenden APDUs zu den entsprechenden Anwendungen. Der Card Manager überwacht u. a. auch, dass die vom Kartenherausgeber festgelegten maximalen Speichergrößen für Application Provider beim Laden ihrer Anwendungen nicht überschritten werden können.

Um alle Informationen über die unterschiedlichen Security Domains, Anwendungen, Lebenszyklusabschnitte der Komponenten und dergleichen auf der Chipkarte zu verwalten, besitzt der Card Manager einen Datenbereich mit Namen Card Registry. Dies ist die zentrale Instanz auf der Chipkarte für die Verwaltung aller Open-Platform-bezogenen Daten.

Entsprechend dem Card Manager ist die Security Domain der informations- und sicherheitstechnische Repräsentant des Application Providers, also des Anbieters der Anwendung. Security Domains dienen zur Bereitstellung von Schlüsseln und kryptografischen

Diensten für eine Applikation unabhängig von denen des Kartenherausgebers. Manche OP-Realisierungen unterstützen das Nachladen von Security Domains. Das Open Platform API (OP API) stellt den Anwendungen Zugriffe auf Verwaltungsdienste des Card Managers zur Verfügung. Die Anwendungen auf der Multapplikation-Chipkarte können nach Open Platform in zwei verschiedene Klassen eingeteilt werden. Die unveränderlichen Anwendungen (*immutable application*) werden bei der Fertigung in den Speicher der Chipkarte geladen und verbleiben dort in konstanter Form. Die veränderlichen Anwendungen (*mutable application*) können während der Fertigung oder im Feld in die Chipkarte geladen, installiert und wieder entfernt werden.

Die OP-Spezifikation beschreibt auch ein generisches Datenformat, um Anwendungen in die Chipkarte zu laden. Die zu übertragenden Daten sind TLV-codiert und bestehen aus den eigentlichen Ladedaten (*load file*) der Anwendung und einem optional vorangestellten DAP-Block (*data authentication pattern*) zur kryptografischen Absicherung der Daten und des Ladevorgangs.

Ist eine Anwendung auf die Chipkarte geladen, dann beginnt der erste Schritt in ihrem Lebenszyklus, welche ebenfalls durch die OP-Spezifikation festgelegt sind. Dieser Schritt heißt installiert (*installed*) und bedeutet, dass die Anwendung richtig gelinkt im zugeteilten

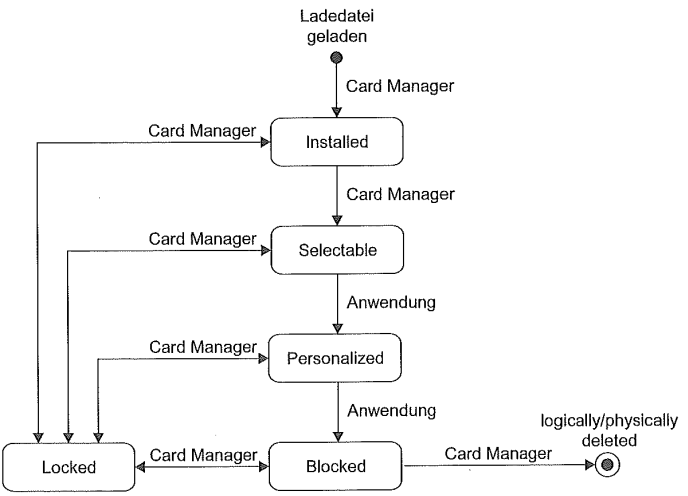


Bild 5.39 Die möglichen Zustände einer Anwendung auf einer Open-Platform-konformen Multiapplication-Chipkarte mit den für die jeweiligen Zustandsübergänge verantwortlichen Instanzen.

Speicher abgelegt ist, sodass sie ausgeführt werden könnte. Sie ist jedoch noch nicht von der äußeren Welt her selektierbar. Dies geschieht im nächsten Zustand, der selektierbar (*selectable*) betitelt ist. Im Folgezustand personalisiert (*personalized*) hat die neu geladene Anwendung nun auch die individuellen bzw. personenbezogenen Daten erhalten. Nun folgt die Zeit des Gebrauchs der Anwendung durch den Kartenbenutzer. Ein möglicher weiterer Zustand ist blockiert (*blocked*), bei dem die Anwendung nicht benutzt werden kann, der jedoch rückgängig zu machen ist. Dieser Zustand würde beispielsweise von der Anwendung

selbst eingenommen, wenn sie ein Sicherheitsproblem diagnostiziert. Ein ähnlicher Zustand wie blockiert ist Sperren (*locked*), dieser ist jedoch irreversibel. Ähnlich verhält es sich mit dem Zustand logisch gelöscht (*logically/physically deleted*), der anzeigt, dass eine bestimmte Anwendung auf der Multiapplication-Chipkarte entweder logisch oder physisch gelöscht wurde. Der mögliche Löschzustand hängt vom jeweiligen Chipkarten-Betriebssystem ab.

Im Zusammenhang mit Open Platform ist noch die Funktion des Delegated Management von Bedeutung. Dabei handelt es sich um die Möglichkeit, dass der Application Provider unabhängig vom Kartenherausgeber (*card issuer*) Anwendungen auf der Chipkarte lädt (*delegated loading*), installiert (*delegated installation*) und löscht (*delegated deletion*). Erreicht werden diese Funktionen, indem die Security Domain des Application Providers diese Privilegien bei ihrer Installation vom Kartenherausgeber erhält.

In der OP-Spezifikation sind eine Reihe von speziellen Kommandos definiert, die für die Funktionen von Open Platform notwendig sind. Einige dieser Kommandos orientieren sich stark an der ISO/IEC 7816-4, sind aber nicht in allen Punkten vollständig übereinstimmend. Die Tabelle 5.21 enthält einen Überblick der OP-Kommandos für Chipkarten sowie jeweils eine dazugehörige Kurzbeschreibung.

Tabelle 5.21 Aufstellung der nach der Open-Platform-Spezifikation festgelegten Chipkarten-Kommandos.

Kommando	Kurzbeschreibung
DELETE	Löschen eines eindeutig identifizierbaren Objekts (z. B. Ladedatei, Anwendung, Schlüssel)
GET DATA	Lesen eines Datenobjekts, in Anlehnung an ISO/IEC 7816-4
GET STATUS	Lesen von Zustandsinformation des Lebenszyklus von Card Manager, Anwendung und Ladedatei. Dies ist das komplementäre Kommando zu SET STATUS.
INSTALL	Installation einer Anwendung durch Aufrufen verschiedener oncard-Funktionen des Card Managers und/oder der Security Domain.
LOAD	Laden einer Anwendung durch Übertragung der Ladedatei
PIN CHANGE/UNBLOCK	Ändern oder Entblocken einer PIN, in Anlehnung an ISO/IEC 7816-4
PUT DATA	Schreiben eines oder mehrerer Datenobjekte, in Anlehnung an ISO/IEC 7816-4
PUT KEY	Schreiben eines oder mehrerer neuer Schlüssel oder Ersetzen bereits vorhandener Schlüssel
SELECT	Auswahl einer Anwendung oder Datei, spezifiziert in ISO/IEC 7816-4
SET STATUS	Schreiben von Zustandsinformation des Lebenszyklus von Card Manager, Anwendung und Ladedatei. Dies ist das komplementäre Kommando zu GET STATUS.

Terminal		Chipkarte
SELECT Card Manager mit AID	→	Returncode := Ergebnis der Selektion
IF (Returncode = ok) THEN Card Manager erfolgreich selektiert ELSE Abbruch	←	*Antwort* [Returncode]
Authentisierung der äußeren Welt gegenüber dem Open Platform Card Manager (z.B. mit EXTERNAL AUTHENTICATE)	→	
IF (Authentisierung = ok) THEN Sequenz fortführen ELSE Abbruch	←	
INSTALL mit Parameter Laden einer Anwendung	→	
REPEAT {		
LOAD mit den Daten der Ladedatei	→	
IF (Returncode = ok) THEN Laden erfolgreich ELSE Abbruch }	←	*Antwort* [Returncode]
UNTIL (Ladedatei vollständig übertragen)		
INSTALL mit Parameter „Installieren einer Anwendung“	→	
INSTALL mit Parameter „Anwendung selektierbar machen“	→	

Bild 5.40 Prinzipieller und überblickshafter Ablauf der Kommandos beim Anlegen einer neuen Anwendung auf einer Chipkarte mit Open Platform

5.12 Nachladbarer Programmcode

Der Abschnitt „nachladbarer Programmcode“ umfasste 1995 in der ersten deutschen Auflage dieses Buches genau 642 Wörter, geschrieben in 65 Zeilen. Der Textumfang zu diesem Thema ist in dieser Auflage mittlerweile auf das 20-fache gestiegen. Allein dieser Punkt zeigt, wie wichtig dieses Thema mittlerweile geworden ist. Man kann wohl ohne zu übertreiben behaupten, dass sich hier innerhalb nur eines Jahres (1997/1998) ein Paradigmenwechsel vollzogen hat. Nachladbarer Programmcode in Chipkarten wird mittlerweile als Regelfall und nicht mehr als Ausnahme angesehen.

Die Gründe, warum das Nachladen von ausführbarem Programmcode so stark an Bedeutung gewonnen hat, sind auch rückblickend nicht ganz eindeutig nachvollziehbar. Eine Triebfeder mag der im Jahr 1994 bekannt gewordene Rechenfehler (FDIV-Befehl) in den damals weit verbreiteten Pentium-Prozessoren gewesen sein. Eine Ausbesserung per Software-Download war nicht möglich, da es ein echter Fehler in der Hardware war. Allerdings gab es für viele Anwendungsprogramme Patches, um den Fehler zu umgehen.

Es ist wahrscheinlich, dass dieser Fehler dazu führte, dass mit geringer zeitlicher Verzögerung einige große Systembetreiber plötzlich die Möglichkeit vorsahen, ausführbaren Programmcode in Chipkarten nachzuladen. Eine der größten Anwendungen mit ausführbarem Programmcode ist die ec-Karte mit Chip in Deutschland. Allerdings wird diese technische Möglichkeit zurzeit nicht genutzt und stellt *de facto* nur einen Rettungsanker für eventuell auftretende schwerwiegende Programmfehler dar. Auch im GSM-System existieren Chipkarten-Betriebssysteme für SIMs, die es ermöglichen, dass Programmcode für spezielle Anwendungen über die Luftschnittstelle nachgeladen werden kann.

Im Gegensatz zu allen anderen Betriebssystemen für Computer ist es aber nicht generell üblich, Programme in Chipkarten nach Ausgabe einzubringen und dort bei Bedarf auszuführen. Dies ist aber eigentlich neben der Datenspeicherung eine der Hauptfunktionen aller Betriebssysteme. Es gibt natürlich Gründe, warum bisher gerade diese Funktionalität in der Chipkartenwelt weitgehend gefehlt hat.

Technisch und funktionell gesehen stellt ausführbarer Programmcode, beispielsweise in Dateien (d. h. EFs) gespeichert, keinerlei Problem dar. Neuere Betriebssysteme bieten deshalb auch die Möglichkeit, Dateien mit ausführbarem Code zu verwalten und auch zu einem Zeitpunkt nach der Personalisierung in die Chipkarte zu laden. Damit ist es möglich, dass beispielsweise ein Anwendungsanbieter Programmcode in der Chipkarte ausführen kann, den der Betriebssystem-Hersteller nicht kennt. So kann ein Anwendungsanbieter einen nur ihm selbst bekannten Verschlüsselungsalgorithmus in die Chipkarte einbringen und dort ausführen. Hierdurch ist es möglich, das Wissen um die Sicherheitsfunktionen des Systems auf verschiedene Parteien zu verteilen, was eine Grundforderung in Sicherheitssystemen ist.

Ein weiterer gewichtiger Grund für den Mechanismus des nachladbaren Programmcodes ist die sich damit eröffnende Möglichkeit der Beseitigung von Programmfehlern (*bug fixing*) in vollständig personalisierten Karten. Erkannte Fehler im Betriebssystem können damit bei bereits ausgegebenen Karten behoben oder zumindest entschärft werden.

Es gibt grundsätzlich zwei Wege, Programmcode in einer Chipkarte auszuführen. Der erste und technisch einfachste Weg ist, kompilierten Code in der Maschinensprache des Zielprozessors (*native code*) in Dateien der Chipkarte zu laden. Dieser Programmcode muss natürlich relokierbar sein, da die Speicheradressen nach außen nicht bekannt sind. Neben der technischen Unkompliziertheit dieser Lösung kann der Programmcode noch mit voller Ausführungsgeschwindigkeit des Prozessors abgearbeitet werden, was diese Lösung gerade für nachladbare Algorithmen sehr interessant macht. Es ist auch weiterhin kein zusätzlicher Programmcode für einen Interpreter in der Chipkarte notwendig. Das große Problem dieser Lösung besteht darin, dass der nachgeladene Programmcode bei Mikrocontrollern ohne MMU (*memory management unit*) auch auf Speicherbereiche von Fremdanwendungen zugreifen kann.

Der zweite Weg, ausführbaren Programmcode in Chipkarten auszuführen, besteht darin, ihn zu interpretieren. Der Interpreter prüft dann während der Programmausführung, welche Speicherbereiche angesprochen werden. Die Interpretation muss aber schnell ablaufen, da ein langsam ausgeführter Programmcode keine Vorteile mehr bringt. Ebenso soll die Implementation eines Interpreters selber so wenig Speicher wie möglich in Anspruch nehmen, da dieser bekanntermaßen stark limitiert ist. Die derzeit bekanntesten Lösungsvarianten dazu sind die Java Card Spezifikation [Javasoft, JCF], der C-Interpreter MEL (*Multos executable language*) von Multos [Maosco] und das mittlerweile wieder eingestellte Windows für Smart Cards. Es gibt für Chipkarten auch seit einigen Jahren einen BASIC-Interpreter [Zeitcontrol]. Interpreter, die dem Programm einer Anwendung einen eigenen geschützten Speicher zur Verfügung stellen, sind im Übrigen nicht für Fehlerbeseitigungen im Betriebssystem von Chipkarten geeignet, da sie auf diese Programm- und Datenteile konsequenterweise keinen Zugriff haben.

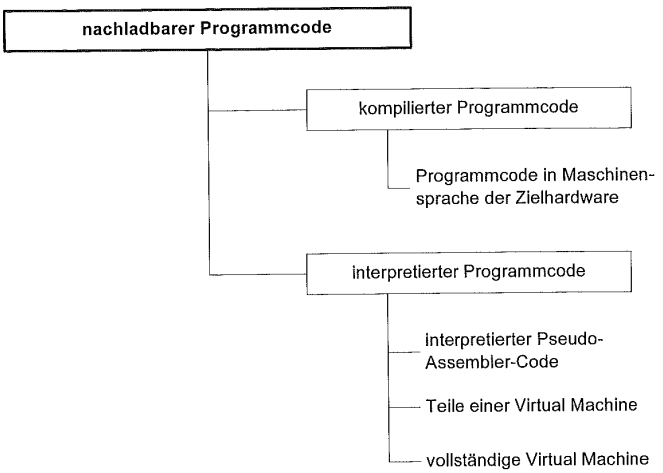


Bild 5.41 Klassifizierungsbaum der Varianten, um ausführbaren Programmcode in Chipkarten-Betriebssysteme nachzuladen und dort auszuführen.

Das Urproblem aller Interpreter ist jedoch die langsame Abarbeitungsgeschwindigkeit, welche für dieses Prinzip immanent ist. Um dieses Minus auszugleichen und um den Programmcode für den eigentlichen Interpreter so klein wie möglich zu halten, gibt es mehrere Lösungsansätze. Die einfachste Methode ist es, einen Pseudocode zu interpretieren, welcher idealerweise den Maschinenbefehlen der Zielhardware möglichst ähnlich sein sollte. Die Abarbeitungsgeschwindigkeit ist durch die maschinennahe Pseudosprache verhältnismäßig hoch, und es kann maschinenunabhängiger Programmcode benutzt werden. Speicherzugriffe während der Interpretation können überwacht werden, was aber nicht zwangsläufig der Fall sein muss. Eine langsamere und programmiertechnisch etwas aufwendigere Lösung ist die Aufspaltung des Interpreters in einen offcard-Teil (*offcard virtual machine*) und einen oncard-Teil (*oncard virtual machine*). Dieser Lösungsweg wird bei vielen heutigen Java Card Implementationen beschritten. Er hat den großen Vorteil eines verlässlichen Speicherschutzes und vollständiger Hardware-Unabhängigkeit. Nachteilig ist die Aufteilung des Interpreters in off- und oncard-Teil. Dies bedingt zwangsläufig einen kryptografischen Schutz beim Übertragen von Programmen zwischen offcard-Teil und oncard-Teil des Interpreters, da mit manipuliertem Programmcode der oncard-Teil des Interpreters bewusst zu einem Fehlverhalten gebracht werden kann.

Die technisch optimale Lösung ist ein vollständiger Interpreter auf der Chipkarte. Damit ist es möglich, in die Chipkarte beliebigen Programmcode zu laden und dort ohne Risiko für andere auf der Chipkarte befindliche Anwendungen auszuführen. Allerdings ist der Programmumfang dazu auch entsprechend groß, weshalb es sicherlich noch einige Jahre und mehrere Chipgenerationen dauern wird, bis diese Variante breiten Einzug in die Chipkartenwelt hält.

5.13 Ausführbarer nativer Programmcode

Mikrocontroller für Chipkarten haben zurzeit noch Prozessoren, die über keinerlei Speicherschutzmechanismen oder Überwachungsmöglichkeiten verfügen. Sobald sich der Programmzähler innerhalb eines fremden Maschinencodes für den Prozessor befindet, liegt die gesamte Kontrolle aller Speicher und Funktionen bei diesem ausführbaren Code. Es gibt dann keinerlei Möglichkeiten mehr, dieses ausführbare Programm in seinen Funktionen zu beschränken. Jede adressierbare Speicheradresse kann unter Umgehung aller Speichermanager oder Handler gelesen und – sofern im RAM oder EEPROM – auch geschrieben werden. Alle Speicherinhalte können dann natürlich auch über die Schnittstelle zum Terminal gesendet werden.

Genau dies ist die Schwachstelle bei ausführbaren und nachladbaren Programmen. Würde man jedermann ein Nachladen von Programmen erlauben oder wäre es durch Umgehung der Schutzmechanismen möglich, ist keine Sicherheit mehr für geheime Schlüssel oder Informationen innerhalb des gesamten Speicherbereichs gegeben. Dies wäre der ideale Angriff auf eine Chipkarte. Diese Karte würde sich nach außen hin wie eine nicht manipulierte Karte verhalten, und mit einem speziellen Kommando könnten der gesamte Speicher ausgelesen oder Teile davon geschrieben werden.

Wäre das Laden nur einigen wenigen Anwendungsanbietern erlaubt – was mit einer gegenseitigen Authentisierung vor dem eigentlichen Laden des Programmcodes ohne weiteres durchführbar ist –, ist das Problem auch nicht aus der Welt geschafft. Der Anwendungsanbieter kann ohne Einschränkungen über die Grenzen seines ihm zugeteilten DFs auf geheime Informationen von anderen vorhandenen Anwendungen zugreifen. Das System wäre wiederum gebrochen.

Doch gibt es noch ein weiteres stichhaltiges Argument gegen von Dritten nachladbaren Programmcode. Um wichtige Funktionen im Betriebssystem nutzen zu können, muss der Ersteller der nachladbaren Datei alle Einsprungadressen und Aufrufparameter kennen. Einige Betriebssystemhersteller erachten es aber für die Sicherheit als relevant, dass so wenig wie möglich über interne Abläufe oder Adressen von Programmcode bekannt ist. Zusätzlich müsste auch noch sichergestellt werden, dass der eingebrachte Code genau das fehlerfrei ausführt, was beabsichtigt ist, und nicht etwa ein trojanisches Pferd enthält. Dies kann dann wiederum nur eine unabhängige Instanz prüfen.

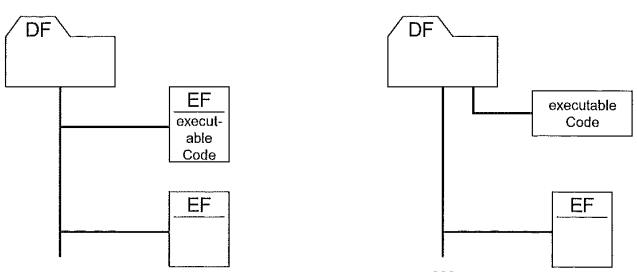


Bild 5.42 Die beiden unterschiedlichen Varianten zur Einbringung von ausführbarem Programmcode in ein übliches Chipkarten-Betriebssystem. Links als ausführbare Datei und rechts als ASC (*application specific commands*).

Die eleganteste und wohl auch zukunftsträchtigste Lösung ist der Einsatz einer Hardware-unterstützten Speicherverwaltung (*memory management unit – MMU*) zusätzlich zum eigentlichen Prozessor in der Chipkarte. Diese prüft den ablaufenden Programmcode mit einer Hardwareschaltung auf die Einhaltung seiner ihm zugewiesenen Grenzen. Erst dann wäre es ohne den Verlust an Sicherheit möglich, jeden Anwendungsbetreiber-Programmcode ohne vorherige Prüfung durch den Kartenherausgeber in die Chipkarte laden zu lassen. Diesem Anwender würde man den physikalisch zusammenhängenden Speicherbereich eines DFs zuweisen. Die MMU prüft die zugeordneten Speichergrenzen während des Aufrufs eines im DF nachgeladenen Programms. Werden die Grenzen überschritten, so kann man über einen Interrupt den Programmablauf unmittelbar stoppen und die Anwendung bis auf weiteres sperren.[10]

Für das Nachladen von nativem Programmcode in Chipkarten existieren zwei Varianten der Realisierung. In der Ersten befindet sich der Programmcode in einem EF mit der Struktur „executable". Nach einer vorherigen Selektion ist das EF mit einem Kommando EXECUTE ausführbar. Je nach Anwendung ist dazu vorher noch eine Authentisierung notwendig. Die Parameter für den Programmaufruf sind im Kommando EXECUTE an die Chipkarte enthalten. Die vom Programm im EF erzeugte Antwort kommt als Teil der Antwort auf das Kommando zum Terminal zurück.

Die zweite Variante gestaltet sich vom Prinzip her etwas anders. Man benutzt dabei einen objektorientierten Ansatz. Dieser ist unter anderem auch in der EN 726-3 als *Application Specific Commands* (ASC) beschrieben. Nach dieser Norm enthält ein DF die vollständige Anwendung mit allen ihren Dateien und anwendungsspezifischen Kommandos. In einem in diesem DF intern vom Betriebssystem verwalteten Speicherbereich kann Programmcode nachgeladen werden. Dies geschieht mit einem speziellen Kommando, das alle dafür notwendigen Informationen an die Chipkarte sendet. Ist nun das betreffende DF selektiert und wird ein Kommando an die Karte gesendet, so prüft das Betriebssystem, ob es zu den Nachgeladenen gehört, und ruft gegebenenfalls ohne weiteres Zutun den im DF befindlichen Programmcode auf. Ist hingegen ein anderes DF selektiert, so existiert das nachgeladene Kommando in diesem Kontext nicht.

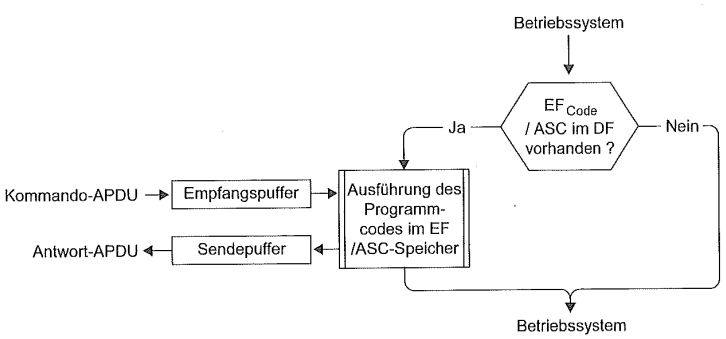


Bild 5.43 Die Grundlage des Aufrufverfahrens für in einem EF gespeicherten ausführbaren Programmcode bzw. Programmen im Rahmen von ASCs (*application specific commands*) in einem üblichen Multiapplication-Betriebssystem für Chipkarten.

[10] Siehe auch Abschnitt 3.4.3 „Zusatzhardware".

Beispiel für in ein EF nachladbaren nativen Programmcode

Es existieren mehrere große Chipkarten-Anwendungen, deren spezifische Betriebssysteme ein Laden von ausführbarem Programmcode nach der Personalisierung vorsehen. Die Spezifikationen dazu sind aber in beinahe allen Fällen vertraulich bzw. zum Teil ist bereits die Tatsache vertraulich, dass Programmcode nachgeladen werden kann. Deshalb sind in diesem Abschnitt die allgemein gültigen Grundlagen unabhängig von einem Betriebssystem aufgeführt und dann nachfolgend eine mögliche Realisierung im Detail dargestellt.

Der nachzuladende Programmcode muss einige Grundvoraussetzungen erfüllen, damit er überhaupt auf der Chipkarte ausgeführt werden kann. So trivial es klingen mag, aber die wichtigste Voraussetzung ist, dass der Prozessortyp (z. B.: 8051, 6805) bekannt ist. Gerade in einer heterogenen Umgebung mit vielen unterschiedlichen Chipkarten-Mikrocontrollern ist die Einhaltung dieser Forderung oft durchaus mit einigem Aufwand verbunden. Einher geht, dass auch das Betriebssystem der Chipkarte einschließlich API (*application programming interface*) mit allen Einsprungadressen, Über- und Rückgabeparametern bekannt sein muss.

Der nachzuladende Programmcode – es handelt sich dabei immer um Maschinencode des Zielprozessors (d. h. Native-Code) – muss entweder so programmiert sein, dass er relokatibel ist, oder die Chipkarte muss ihn während des Ladens on-the-fly selber relokieren. Die Forderung nach Relokatierbarkeit (d. h. Verschiebbarkeit des Programmcodes im Speicher) muss deshalb aufgestellt werden, weil die Speicheradresse für die Programmablage nur das Chipkarten-Betriebssystem kennt und der äußeren Welt unbekannt ist. In der Regel wird der Programmcode bereits bei der Softwareentwicklung so erstellt, dass er relokierbar ist. Dies bedeutet konkret, dass beispielsweise in diesem Programm keine Sprünge an absolute physikalische Adressen gemacht werden dürfen, sondern lediglich Sprünge relativ zur Adresse des Sprungbefehls.

Erfüllt der Programmcode alle diese Voraussetzungen, kann er prinzipiell in den Speicher einer Chipkarte geladen und dort ausgeführt werden. Der Programmcode kann natürlich je nach Bedarf aufgebaut sein. Eine mögliche Struktur zeigt das folgende Bild, wobei diese aber je nach Betriebssystem völlig verschieden sein kann. Das erste Datenelement in dem Beispiel ist ein eindeutiges Kennzeichen für das Chipkarten-Betriebssystem, dass es sich um Programmcode handelt. Allgemein bekannt ist so ein Kennzeichen auch als „Magic Number". Diese kurze Bytesequenz setzt sich beispielsweise bei Java-Class-Dateien aus den vier Bytes 'CAFEBABE' zusammen.

Im Anschluss an das Kennzeichen folgt bereits der Programmcode, welcher in diesem Beispiel nochmals in vier Teile aufgegliedert ist. Der erste Teil ist für alle notwendigen Initialisierungen, Sicherung von Daten und Ähnliches vorgesehen. Nach dieser Startup-Routine folgt die eigentliche Funktionsroutine, welche den Programmcode für die gewünschte Aufgabe enthält. Daran schließt sich das Pendant zur Startup-Routine an, die Shutdown-Routine. Diese stellt sicher, dass das Programm korrekt abgeschlossen wird, und führt dazu nach Bedarf eine Rücksicherung von Daten oder ggf. Änderungen auf dem Stack durch.

Optional und am Schluss dieser drei Programmteile befindet sich das vierte Programmelement. Es kann Programmcode enthalten, der resistent in die Software der Chipkarte eingebunden werden soll. Typischerweise würden hier Bugfixes für das Chipkarten-Betriebssystem abgelegt. Die vorangehenden drei Routinen würden Zeiger oder Handles so verändern, dass diese Programmroutinen permanent in die Programmabläufe des Betriebssystems eingebunden sind. Das Ganze verhält sich sehr ähnlich wie die noch aus DOS-Zeiten bekannten

TSR-Programme (*terminate and stay resistant*), welche sich nach einem einmaligen Aufruf bis zum nächsten Reset fest im Betriebssystem verankerten. Die resistenten Programme in diesem Fall wären jedoch nach einmaligem Aufruf auf Dauer fest installiert und nicht nur für eine Sitzung.

Es wird hier davon ausgegangen, dass das nachgeladene Programm mit einem CALL-Befehl aufgerufen wird und mit einem RETURN-Befehl zum Aufrufer zurückkehrt. Prinzipiell wäre auch ein direkter Sprung mit JUMP auf den ersten Maschinenbefehl möglich, dies hätte aber den Nachteil, dass dann dem aufgerufenen Programm nicht bekannt ist, von wem es aufgerufen wurde.

Zur Absicherung gegen unbeabsichtigte Veränderung sollte der gesamte Datenblock noch mit einem Fehlererkennungscode (*error detection code – EDC*) abgesichert sein. Alternativ dazu würde sich hier natürlich auch eine digitale Signatur als zusätzlicher Schutz anbieten. Die Chipkarte würde dann den öffentlichen Schlüssel besitzen und der Ersteller des Programmcodes den dazugehörigen geheimen Schlüssel. Damit wäre bindend sichergestellt, dass authentischer Programmcode auf der Chipkarte gestartet werden kann.

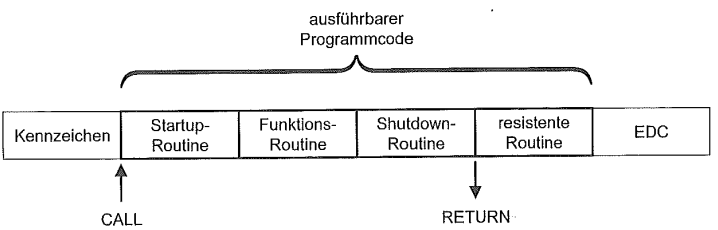


Bild 5.44 Möglicher Aufbau eines nativen und in ein EF nachladbaren und ausführbaren Programmcodes.

Der nachgeladene Programmcode kann entweder in einem EF gespeichert werden oder in einem für die äußere Welt unsichtbaren Speicherbereich für Programme innerhalb eines DFs. Im Folgenden wird die erste Variante genauer aufgezeigt, da diese in der Praxis wesentlich häufiger anzutreffen ist.

Ideal zur Ablage von Programmcode sind EFs mit der Struktur „transparent" geeignet, da diese sich zweckmäßig mit dem UPDATE BINARY-Kommando und Offsetangabe in mehreren Teilstücken beschreiben lassen. Zudem ist ihre maximale Größe von über 65 kByte selbst für umfangreiche Programme mehr als ausreichend. Diese EFs können die Eigenschaft „executable" haben, sodass in ihnen gespeicherter Programmcode direkt mit dem Kommando EXECUTE aufgerufen wird.

Manche Betriebssysteme besitzen aber auch eine von „transparent" abgeleitete Dateistruktur, welche sich dann „execute" nennt. Für die äußere Welt ist dies nicht von besonderer Bedeutung, zumal in der Regel auf beide Varianten mit UPDATE BINARY und EXECUTE zugegriffen werden kann. Mit dem FID bzw. Short-FID kann das EF selektiert werden. Die Zugriffsbedingung zum Lesen ist grundsätzlich auf „never" gesetzt. Das Schreiben von Daten ist in der Regel nur nach vorheriger Authentisierung und mit Secure Messaging erlaubt.

Der Ablauf in Bild 5.45 zeigt im Überblick, wie Programmcode auf sichere Weise in das EF einer Chipkarte geladen werden kann. Sollte kein entsprechendes EF vorhanden sein, muss dieses vorher erzeugt werden. Im Ablaufbild 5.46 ist überblickshaft dargestellt, dass das EF zuerst selektiert und anschließend der Programmcode mit dem Kommando

EXECUTE gestartet werden muss. Das Kommando sieht optional eine Datenübergabe im Kommando-Body vor. Analog verhält es sich mit der Antwort, in welcher Daten bei Bedarf an das Terminal zurückgegeben werden können. Das aufgerufene Programm muss dazu natürlich die Daten aus dem Empfangspuffer lesen, auswerten und dann die Antwort in den Sendepuffer zurückschreiben.

Hintergrundsystem		**Chipkarte**
selektiere ein EF mit der Struktur execute	→	
gegenseitige Authentisierung von Hintergrundsystem und Chipkarte	→	
schalte Secure Messaging ein		
sende n · UPDATE BINARY-Kommando mit dem Datenteil als ausführbaren Programmcode, geschützt mit Secure Messaging	→	

Bild 5.45 Ein möglicher Ablauf beim Laden von ausführbarem Programmcode in ein bereits vorhandenes EF mit der Struktur „execute". Die Zugriffsbedingung für UPDATE BINARY ist eine gegenseitige Authentisierung zwischen Chipkarte und Terminal sowie die Datenübertragung mit Secure Messaging.

Hintergrundsystem		**Chipkarte**
Selektiere EF mit ausführbarem Programmcode	→	
Sende EXECUTE-Kommando	→	Prüfe Kennzeichen (*magic number*) des Programms Prüfe EDC des Programms Rufe den ersten Maschinenbefehl mit CALL auf

Bild 5.46 Ein möglicher Ablauf beim Starten von ausführbarem Programmcode. Dieser ist in diesem Beispiel in einem EF mit der Struktur „execute" abgelegt.

Aufgrund der hohen Anforderungen betreffend eine eindeutige Identifizierung von Mikrocontroller, Betriebssystem und interner Software-Schnittstellen sowie des System-Managements ist es üblich, den Download von Programmen nur online zu einem Hintergrundsystem durchzuführen. Die dort befindlichen Datenbanken enthalten entweder alle notwendigen Informationen in Abhängigkeit von einer eindeutigen Chipnummer oder erhalten diese Informationen online in einer Ende-zu-Ende-Verbindung direkt von der Chipkarte. In Abhängigkeit davon wird dann ein vorhandenes Programm mit der gewünschten Funktionalität ausgewählt und mit den geforderten Sicherheitsmechanismen zur Chipkarte übertragen. Die geheimen Schlüssel dazu werden im Hintergrundsystem üblicherweise ausschließlich innerhalb eines Sicherheitsmoduls verwaltet und benutzt. Den üblichen Ablauf zeigen wir im nachfolgenden Bild nochmals auf.

Die oben beschriebene Art und Weise der Einbringung von nativem Programmcode in Chipkarten besitzt einige für die Praxis attraktive Vorteile. Das Verfahren ist unkompliziert, robust und mit geringem Aufwand an Programmcode in einem Chipkarten-Betriebssystem realisierbar. Der ausgeführte Programmcode muss nicht interpretiert, sondern kann direkt vom Prozessor abgearbeitet werden. Daraus resultiert eine hohe Ausführungsgeschwindigkeit sowie die Möglichkeit, auch komplexe Algorithmen (z. B.: DES, IDEA o. Ä.) mit diesem Verfahren nachzuladen.

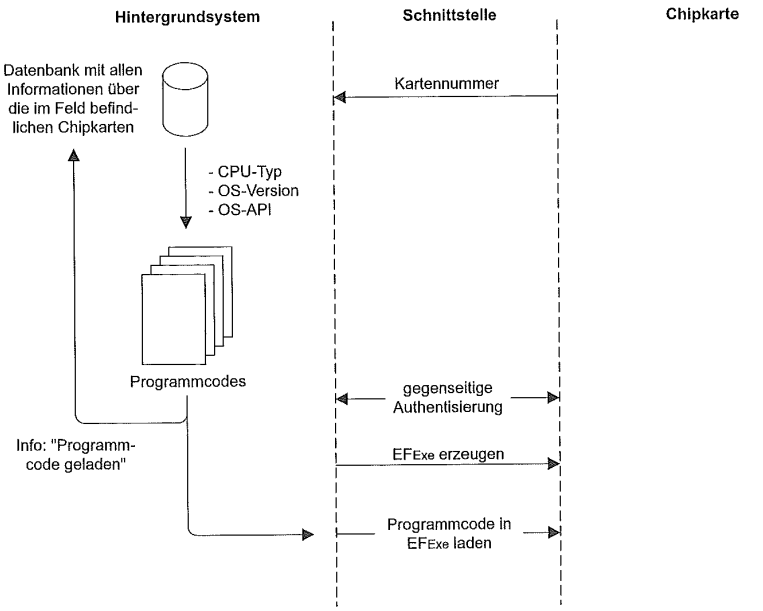


Bild 5.47 Ablauf beim online-Nachladen von nativen Programmcodes in EFs einer Chipkarte von einem Hintergrundsystem aus. Es kann bei diesem Schema abhängig vom Chipkarten-Betriebssystem und der Mikrocontroller-Hardware unterschiedlicher Programmcode zur Chipkarte übertragen werden. Der dargestellte Ablauf könnte beispielsweise sehr gut in einer GSM-Anwendung realisiert werden.

Interpreter-basierte Systeme können dies durch die geringe Abarbeitungsgeschwindigkeit des Programmcodes auch auf absehbare Zeit nicht leisten. Solange keine Hardware-basierte Speicherverwaltung (MMU) die wahlfreien Speicherzugriffe einschränkt, kann dieses Verfahren hervorragend für die Behebung von Fehlern in der Chipkarten-Software nach Kartenausgabe genutzt werden. Dies ist für den Fall des Falles eine Hintertür, die einzig und allein durch diese Art der Softwarenachladung erreichbar ist, da beispielsweise Technologien wie Java auf Chipkarten eine strikte und unbedingte Speicherseparierung verwirklichen. Ist eine MMU vorhanden, dann könnte ggf. immer noch über einen Administrator-Modus die Speicherüberwachung vorübergehend deaktiviert werden.

Somit sind wir schon bei den Nachteilen. Das Nachladen von ausführbaren nativen Programmen setzt großes Wissen um die eingesetzte Hardware bzw. das Betriebssystem der Chipkarte voraus. Unter Umständen muss für jede dieser Varianten ein eigenes Programm gleicher Funktionalität vorgehalten werden. Die zweite große Schattenseite dieser Methode ist, dass sicherheitstechnisch die Notwendigkeit besteht, dass nur der Kartenherausgeber den Programmcode erstellen oder erstellen lassen kann. Es muss strikt verboten sein, fremde und unbekannte Programme in die Chipkarte zu laden, da diese ab dem Start das Kommando über den Mikrocontroller ohne weitere Reglementierungsmöglichkeiten erhalten. Sie könnten dann beispielsweise die geheimen Schlüssel der anderen auf der Karte befindlichen Anwendungen auslesen und über die I/O-Schnittstelle zum Terminal senden.

Einen schwachen Schutz vor Angriffen über diesen Mechanismus stellen Evaluierungen des Programmcodes durch den Kartenherausgeber dar. Besser jedoch ist in diesem Fall eine Hardware-basierte Speicherverwaltung, die dem nachgeladenen Programm nur einen bestimmten Bereich im RAM und EEPROM zur Verfügung stellt und bei dem Versuch der Überschreitung das gestartete Programm sofort beendet.[11] Damit können die Einzelnen auf der Chipkarte befindlichen Anwendungen vollständig voneinander abgeschottet werden. Zurzeit behilft man sich jedoch noch mangels entsprechender MMUs mit der Prüfung der nachzuladenden Programme.

5.14 Offene Plattformen

Mit der Verbreitung von Java Card, Multos und Windows for Smart Cards verbreitete sich der Begriff der offenen Plattform (*open platform*). Darunter versteht man Chipkarten-Betriebssysteme, die unabhängig vom Betriebssystemhersteller die Möglichkeit bieten, dass Dritte Anwendungen und Programme auf die Chipkarte laden können. Die Spezifikationen von offenen Plattformen sind in der Regel meist öffentlich und werden innerhalb eines Firmenkonsortiums (z. B. Java Card Forum) erstellt. Offene Chipkarten-Betriebssysteme werden meist von mehreren Herstellern mit ähnlicher bzw. mit zueinander kompatibler Funktionalität angeboten.

Das Gegenteil von offenen Plattformen sind so genannte proprietäre Plattformen, welche in häufig abwertender Form für firmenspezifische Lösungen benutzt wird. Die dazugehörigen Spezifikationen sind dabei oftmals nicht vollständig veröffentlicht oder Eigentum einer einzelnen Firma.

Sowohl die Verwendung des Begriffs offen als auch des Begriffs proprietär ist aber keinesfalls eindeutig und unparteiisch, sondern in vielen Fällen stark Marketing-getrieben. Objektiv gesehen ist so manches so genannte offene Chipkarten-Betriebssystem beachtlich proprietär und abhängig von einer bestimmten Firma. Eine echte offene Plattform im Sinne von Linux mit freiem Zugang zum Sourcecode, Lizenzfreiheit und Unabhängigkeit von bestimmten Firmen oder Instanzen gibt es zurzeit bei Chipkarten-Betriebssystemen nicht.

5.14.1 Java Card

Im Jahr 1996 wurde von Europay ein Papier über OTA (*Open Terminal Architecture*) vorgestellt, in welchem ein Forth Interpreter für Terminals beschrieben und in weiten Teilen spezifiziert war. Der Zweck war, eine einheitliche Softwarearchitektur für Terminals zu schaffen, um die Grundlage für eine Hardware-unabhängige Terminalprogrammierung zu schaffen. Dann müsste man eine bestimmte Anwendung (z. B. Bezahlen mit Kreditkarte) nur noch ein einziges Mal programmieren, und diese Software würde auf allen Terminals der verschiedenen Hersteller unverändert laufen. Dieses Modell wurde aber bisher nie vollständig realisiert, es sorgte allerdings in der Chipkartenwelt für ausführliche Diskussionen.

[11] Siehe auch Abschnitt 3.4.3 „Zusatzhardware".

Als dann im Herbst 1996 bekannt wurde, dass die Firma Schlumberger eine Chipkarte entwickelt, welche plattformunabhängig Programme abarbeiten kann, die in der Programmiersprache Java erstellt sind, hielt sich das Erstaunen in Grenzen. Das Prinzip eines Interpreters auf speicherplatzarmen Mikrocontrollern war durch die Open Terminal Architecture (OTA) schon reichlich bekannt. Die veröffentlichte Spezifikation Java Card 1.0 sah zur Einbindung von Java in das ISO/IEC 7816-4 Betriebssystem ein dazugehöriges API (*application programming interface*) vor, sodass von Java aus auf das bei Chipkarten übliche Dateisystem mit MF, DFs und EFs zugegriffen werden konnte.

Nach kurzzeitiger Verwunderung vieler Hersteller von Chipkarten-Betriebssystemen, warum eine Sprache wie Java, die einen üblichen Speicherbedarf weit jenseits von einem Megabyte hat, auf Chipkarten verwendet werden soll, kam es aber bereits im Frühjahr 1997 zu einem ersten Treffen von nahezu allen großen Chipkartenherstellern und der Firma Sun, die Java entwickelt und bekannt gemacht hat.

Dies war die erste Konferenz des inzwischen so genannten Java Card Forums (JCF), welches das international tätige Standardisierungsgremium für Java auf Chipkarten ist. Die Aufgabe der technischen Gruppe des Java Card Forums ist es, eine Untermenge von Java für Chipkarten festzulegen, den Rahmen für den Java Interpreter (d. h. die *Java Virtual Machine – JVM*) zu spezifizieren und sowohl ein allgemeines als auch anwendungsspezifische (z. B.: Telekommunikation, Zahlungsverkehr) APIs als Schnittstelle zwischen Chipkarten-Betriebssystem und Java festzulegen. Die Marketing-Gruppe des Java Card Forums hat die Aufgabe, die Technologie von Java auf Chipkarten zu fördern.

Die zurzeit[12] aktuellen Spezifikationen haben die Namen Java Card Virtual Machine Specification, Java Card Runtime Environment (JCRE) Specification und Java Card Application Programming Interface. Sie sind auch in der jeweils aktuellen Version auf dem WWW-Server des Java Card Forums kostenlos verfügbar [JCF].

Die Programmiersprache Java

Im Jahr 1990 begann eine Entwicklungsgruppe bei der Firma Sun um James Gosling damit, eine neue Programmiersprache zu entwickeln. Ziel sollte eine Hardware-unabhängige, sichere und moderne Sprache sein, für den Einsatz bei Mikrocontrollern im Konsumerbereich (z. B. Toaster, Espressomaschine). Dort existiert eine große Typenvielfalt an Mikrocontrollern unterschiedlicher Architektur und mit zusätzlichen häufigen Hardware-Änderungen, was es für Entwickler ziemlich schwer macht, portablen Programmcode zu schreiben. Bemerkenswerterweise entsprechen Chipkarten exakt diesem ursprünglich ins Auge gefassten Einsatzgebiet.

Die Sprache Oak, benannt nach der Eiche vor James Goslings Büro, wurde 1995 in Java[13] umbenannt und die damit zu erreichenden Ziele neu festgelegt. Ab Sommer 1995 wurde Java von der Firma Sun mit großem Aufwand als die Hardware-unabhängige Sprache für das heterogene World Wide Web Internet propagiert. Der damals oft genannte Werbespruch von Sun „Write Once – Run Anywhere" zeigt wohl am deutlichsten den Grad der angestrebten Hardware-unabhängigkeit.

---

[12] d. h. im Sommer 2002

[13] Java steht in diesem Zusammenhang für den amerikanischen Slang-Ausdruck „Tasse Kaffee" und weder für die Insel im Pazifik noch für das ebenfalls gleichnamige französische Kleingebäck.

Der Beginn der Verbreitung von Java fällt mit der Zeit zusammen, als das enorme Wachstum des WWW begann.[14] Aus den unterschiedlichsten Gründen, die jedoch nicht nur technischer Natur waren, sondern durchaus in den Bereich Geschäftspolitik und Weltanschauung gehören, begeisterten sich weltweit Entwickler, Universitäten und Softwarefirmen für die neue Sprache [Franz 98]. Dies führte dazu, dass Java in enorm kurzer Zeit zum De-facto-Standard für Internetanwendungen wurde. Begünstigt wurde dies natürlich auch durch die Eigenschaften dieser neuen Sprache.

Die Programmiersprache Java ist eine vollständig objektorientierte und stark typisierte Sprache. Sie ist für Softwareentwickler leicht zu erlernen, da es viele Gemeinsamkeiten mit C und C++ gibt. Java ist auch eine robuste Sprache, welche nicht alle Tricks und gerne benutzten Unsauberkeiten zulässt, die beispielsweise in C/C++ erlaubt sind. So existieren in Java keine Zeiger, die Grenzen von Feldern werden zur Laufzeit überwacht, und es findet eine strikte Typprüfung statt. Zusätzlich wird die Speicherverwaltung von Java und einem dazugehörigen Garbage Collector übernommen, sodass die bei C/C++ so gefürchteten Speicherlecks schon im Ansatz unmöglich gemacht werden. Java ist auch eine sichere Programmiersprache, was bedeutet, dass bei ausgeführten Programmen während der Ausführung geprüft wird, welche Funktionen sie ausüben wollen, und von der Laufzeitumgebung dabei nach Bedarf gestoppt werden können. Dies ist dann einer von mehreren möglichen Gründen für den Aufruf einer so genannten Ausnahmebehandlung (*exception*). Tritt eine Exception auf, wird ein bestimmter Programmteil aufgerufen, in dem dann die für diesen Fall angebrachte Reaktion festgelegt werden kann.[15]

Ein Großteil dieser Eigenschaften wird erst möglich gemacht, weil Java eine interpretierte Programmiersprache ist und nicht direkt vom Zielprozessor ausgeführt wird. Neben diesen Eigenschaften besitzt Java noch einige weitere, wie die Fähigkeit zu Multithreading und verteilter Ausführung von Programmen, doch wird dies zurzeit im Chipkarten-Umfeld nicht unterstützt.

Es wurde von der Firma Sun beabsichtigt, die Programmiersprache Java als internationale ISO-Norm zu standardisieren. Dies ist jedoch nicht gelungen, wobei die Gründe dafür mehrschichtig sind. Ein Grund war sicherlich, dass die Revisionszyklen von ISO mit fünf Jahren zu lang für eine neue Programmiersprache wie Java sind und aktuell notwendige Anpassungen damit sehr schwierig und zu langfristig geworden wären. Ein weiterer vorstellbarer Grund dürfte gewesen sein, dass die Verwendung der Java-Technologie in Produkten, wie beispielsweise in Chipkarten, einen Lizenzvertrag mit der Firma Sun voraussetzt, der mit erheblichen Kosten verbunden ist. Dies widerspricht den üblichen Konventionen von ISO-Normen.

Die Eigenschaften von Java

Programme, geschrieben in Java, werden von einem Compiler in den Java Bytecode übersetzt, der nichts anderes ist als ein Prozessor-unabhängiger Objektcode. Der Bytecode ist sozusagen ein Programm, bestehend aus Maschinenbefehlen, für einen virtuellen Java-Prozessor. Dieser Prozessor existiert nicht wirklich, sondern wird vom jeweiligen Zielprozessor simuliert. Diese Simulation geschieht in der Java Virtual Machine (JVM oder VM),

[14] Im Jahr 1993 gab es weltweit nur vier WWW-Server!

[15] Siehe auch [Arnold 00].