HANSER

Leseprobe

Wolfgang Rankl, Wolfgang Effing

Handbuch der Chipkarten

Aufbau - Funktionsweise - Einsatz von Smart Cards

ISBN: 978-3-446-40402-1

Weitere Informationen oder Bestellungen unter

http://www.hanser.de/978-3-446-40402-1

sowie im Buchhandel.

Kapitel 8

Kommunikation mit Chipkarten

Voraussetzung für alle Interaktionen zwischen Chipkarte und Terminal ist die Möglichkeit der Kommunikation miteinander. Dafür steht aber im Falle der klassischen ISO/IEC-Protokolle T=0 und T=1 nur eine einzige Leitung zur Verfügung. Auf Basis dieser elektrischen Verbindung werden Informationen zwischen Karte und Terminal in digitaler Form ausgetauscht. Da es nur diese eine Leitung gibt, können Terminal und Karte nur wechselseitig senden, der jeweils andere Teilnehmer muss zu diesem Zeitpunkt empfangen. Dieses abwechselnde Senden und Empfangen wird als Halbduplex-Verfahren bezeichnet.

Das Vollduplex-Verfahren, bei dem beide Teilnehmer gleichzeitig senden und empfangen können, ist zur Zeit in der Chipkartenwelt nicht verwirklicht. Da jedoch viele Chipkarten-Mikrocontroller über zwei I/O-Ports verfügen und zwei der acht Kontaktflächen für zukünftige Anwendungen (wie beispielsweise eine zweite I/O-Verbindung oder USB) reserviert sind, wäre eine Vollduplex-Verbindung technisch möglich.

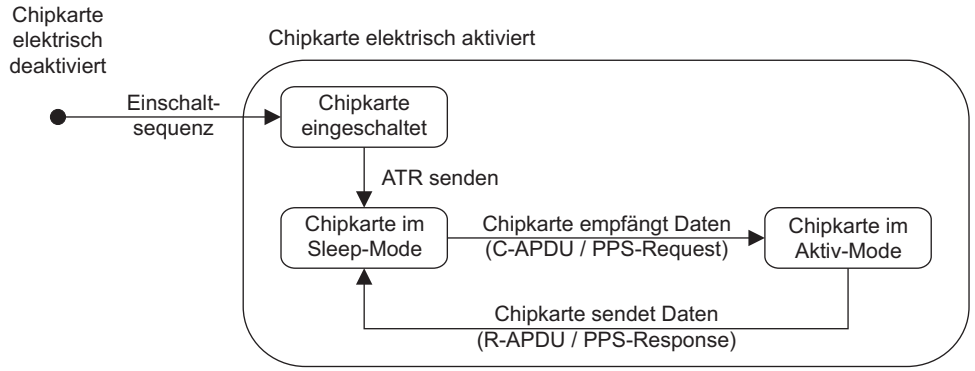

Bild 8.1 Die generellen Zustände einer Chipkarte beim Anschalten und während der Kommunikation mit dem Terminal.

Die Kommunikation mit der Chipkarte wird immer vom Terminal angestoßen, die Karte reagiert immer nur auf die Kommandos des Terminals. Sie sendet also nie ohne äußeren Anstoß Daten. Daraus ergibt sich ein reines Master-Slave-Verhalten, mit dem Terminal als

Master und der Karte als Slave. Auch das bei Telekommunikations-Chipkarten eingesetzte Verfahren der proaktiven Kommandos,[1] bei dem die Chipkarte Kommandos an das Terminal senden kann, baut auf dieser Zuordnung von Master und Slave auf. Sobald die Chipkarte das an sie gesendete Kommando abgearbeitet und die Antwort an das Terminal gesendet hat, fällt sie in den stromsparenden Sleep-Modus, aus dem sie nur zur Ausführung eines Kommandos aufwacht. Bild 8.1 auf der vorherigen Seite stellt dies als Zustandsdiagramm dar.

Nachdem eine Chipkarte in ein Terminal gesteckt wurde, werden als Erstes die Kontakte der Karte mit denen des Terminals verbunden. Danach werden die fünf belegten Kontakte in der richtigen Reihenfolge elektrisch aktiviert.[2] Die Chipkarte führt daraufhin automatisch einen Power-On-Reset aus und sendet einen Answer to Reset (ATR) zum Terminal. Dieses wertet den ATR, der diverse Karten- und Übertragungsparameter anzeigt, aus und schickt danach ein erstes Kommando. Die Chipkarte bearbeitet das Kommando und erzeugt eine Antwort, die sie zum Terminal zurücksendet. Bild 8.2 zeigt dieses Ping-Pong-Spiel von Kommando und Antwort, das sich bis zum Deaktivieren der Chipkarte fortsetzt.

Zwischen ATR und dem ersten Kommando an die Chipkarte kann vom Terminal noch ein Protocol Parameter Select (PPS)-Kommando gesendet werden. Mit diesem Kommando, das ebenso wie der ATR unabhängig vom Übertragungsprotokoll ist, kann das Terminal verschiedene Übertragungsparameter des Protokolls der Karte einstellen.

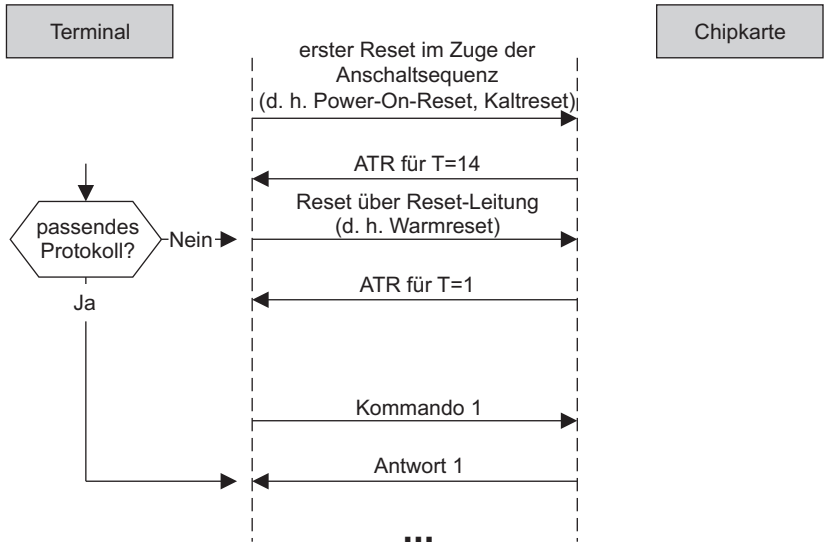

Bild 8.2 Answer to Reset (ATR), Protocol Parameter Selection (PPS) und das erste Kommando-Antwort-Paar bei der Datenübertragung zwischen Chipkarte und Terminal.

Die gesamte Datenübertragung von und zur Chipkarte kann im Rahmen des OSI-Schichtenmodells dargestellt werden. Dabei wird zwischen den elektrischen Geschehnissen auf der I/O-Leitung, den logischen Abläufen der eigentlichen Übertragungsprotokolle und dem Verhalten der darauf aufbauenden Anwendung unterschieden. Das Verhalten und die Abläufe in

[1] Siehe auch Abschnitt 19.4.4 „Die SIM“ auf Seite 879.
[2] Siehe auch Abschnitt 4.6 „An-/Abschaltsequenz“ auf Seite 75.

und zwischen diesen Schichten ist dabei in mehreren internationalen Normen festgelegt. Die Zusammenhänge sind in Bild 8.3 illustriert.

Die im Folgenden vorgestellten asynchronen Übertragungsprotokolle sind in ihrer Funktionalität jeweils nach der entsprechenden Norm beschrieben. Es sind alle Parameter und Einstellungen angegeben, die im Rahmen des Protokolls möglich sind. In der Praxis kommt es aber vor, dass Chipkarten nicht alle Varianten der Übertragungsprotokolle unterstützen, da der verfügbare Speicherplatz dafür nicht ausreicht.

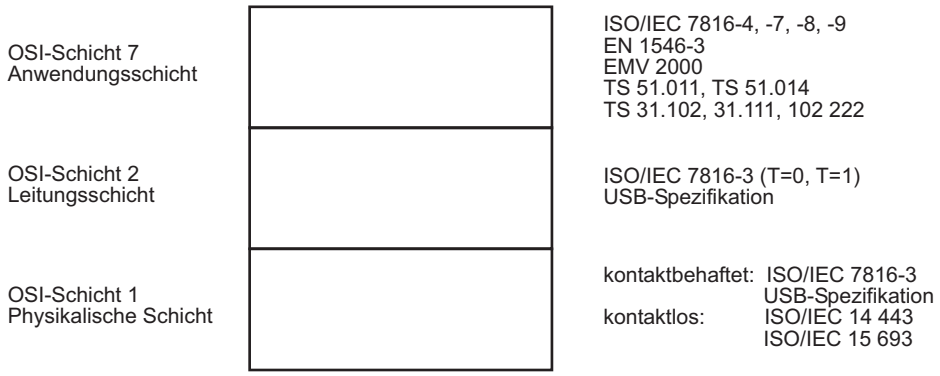

Bild 8.3 Das OSI-Modell der Kommunikation zwischen Terminal und Chipkarte und die wichtigsten dazugehörigen Normen im Überblick.

Funktional gesehen kann man die Variationsmöglichkeiten nur als ein Angebot von Varianten sehen, aus dem man für die Anwendung bzw. Chipkarte das Optimale heraussucht. Wichtig dabei ist die Tatsache, dass die ausgewählten Parameter nicht exotisch sind, so dass die betreffenden Chipkarten mit möglichst allen Terminals kommunizieren können.

Im Terminalbereich verhält es sich bei den Übertragungsprotokollen etwas anders. Dort ist üblicherweise die vollständige Funktionalität der jeweiligen Norm implementiert, da genügend Speicher vorhanden ist.

Die oben aufgezeigten Abläufe sind für die beiden Übertragungsprotokolle T=0 und T=1 gültig, welche beinahe ausschließlich bei Chipkarten eingesetzt werden. Neue Protokolle wie USB[1], MMC[2] oder SWP[3] benutzen angepasste Abläufe und Protokollelemente, welche in den entsprechenden Abschnitten im Detail beschrieben sind.

8.1 Answer to Reset (ATR)

Nach dem Anlegen der Spannungsversorgung, des Taktes und des Resetsignals sendet die Chipkarte am I/O-Pin einen Answer to Reset (ATR) aus. Dieser maximal 33 Byte lange Datenstring wird laut Norm ISO/IEC 7816-3 immer mit dem Teiler (*clock rate conversion factor*) 372 gesendet und enthält verschiedene Informationen über die Datenübertragung

[1] Siehe auch Abschnitt 9.4 „Übertragungsprotokoll USB“ auf Seite 295.
[2] Siehe auch Abschnitt 9.5 „Übertragungsprotokoll MMC“ auf Seite 301.
[3] Siehe auch Abschnitt 9.6 „Single-Wire-Protokoll“ auf Seite 302.

und die Chipkarte. Dieser Teiler ist auch dann zu verwenden, wenn das auf den ATR folgende Übertragungsprotokoll einen anderen Teiler (z. B. 64) benutzt. Damit ist sichergestellt, dass der ATR jeder Chipkarte unabhängig von den Parametern des Übertragungsprotokolls empfangen werden kann.

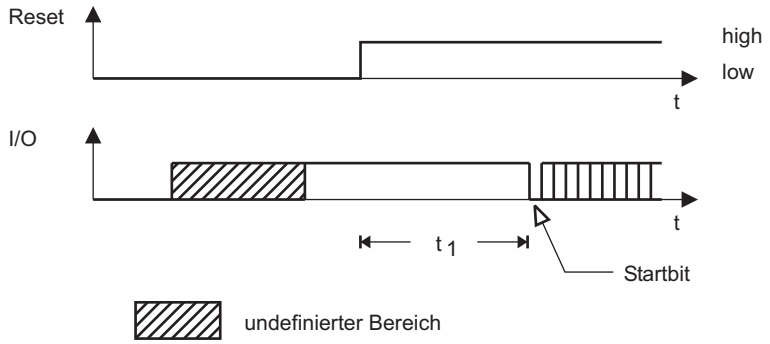

Bild 8.4 Zeitlicher Ablauf des Resetsignals und Start des ATRs nach ISO/IEC 7816-3 mit der zeitlichen Bedingung 400 Takte $\leq t_1 \leq$ 40 000 Takte.

Tabelle 8.1 Die Datenelemente des ATR und ihre Bezeichnungen nach ISO/IEC 7816-3.

Datenelement	**Bezeichnung**
TS	The Initial Character
T0	The Format Character
TA_1, TB_1, TC_1, TD_1, ...	The Interface Characters
T1, T2, ..., TK	The Historical Characters
TCK	The Check Character

Es kommt selten vor, dass ein ATR die maximale Länge erreicht. Viel öfter ist der ATR nur aus wenigen Bytes aufgebaut. Gerade bei Anwendungen, in denen die Karte sehr schnell nach der Anschaltsequenz benutzbar sein muss, darf der ATR nicht zu lang sein. Ein typisches Beispiel hierzu wäre die Bezahlung von Maut an Autobahnen mit elektronischen Geldbörsen in Form von Chipkarten. Trotz hoher Geschwindigkeit der Fahrzeuge beim Durchfahren der Erfassungsstellen muss in der kurzen zur Verfügung stehenden Zeit nach dem Anschalten der Chipkarte ein zuverlässiges Abbuchen möglich sein.

Mit dem Aussenden des ATR muss zwischen 400 und 40 000 Taktzyklen nach dem Freigeben des Resetsignals durch das Terminal begonnen werden. Dies entspricht bei einer Taktfrequenz von 3,5712 MHz der Zeitspanne von 112 μs bis 11,20 ms, und bei 4,9152 MHz sind es 81,38 μs bis 8,14 ms. Empfängt das Terminal während dieses Zeitraums nicht den Beginn des ATRs, so wiederholt es üblicherweise dreimal die Anschaltsequenz und versucht dabei, einen ATR zu detektieren. Schlagen alle diese Versuche fehl, so nimmt das Terminal die Karte als fehlerhaft an und reagiert entsprechend.

Während des ATRs darf zwischen den Startflanken der einzelnen Bytes nach ISO/IEC 7816-3 eine Zeitdauer von 9 600 etu liegen. Diese Zeit hat die Bezeichnung initial waiting time. Bei einem Takt von 3,5712 MHz ist dies eine Zeitdauer von genau einer Sekunde. Dies be-

deutet, dass es nach der Norm erlaubt ist, zwischen den einzelnen Bytes im ATR jeweils eine ganze Sekunde zu warten und erst dann das nächste Byte zum Terminal zu senden. In einigen Chipkarten-Betriebssystemen wird diese Zeit für interne Berechnungen und EEPROM-Schreibzugriffe benutzt. Oft findet dabei auch die Bereinigung der internen Schreibpuffer für atomare Abläufe statt.[1]

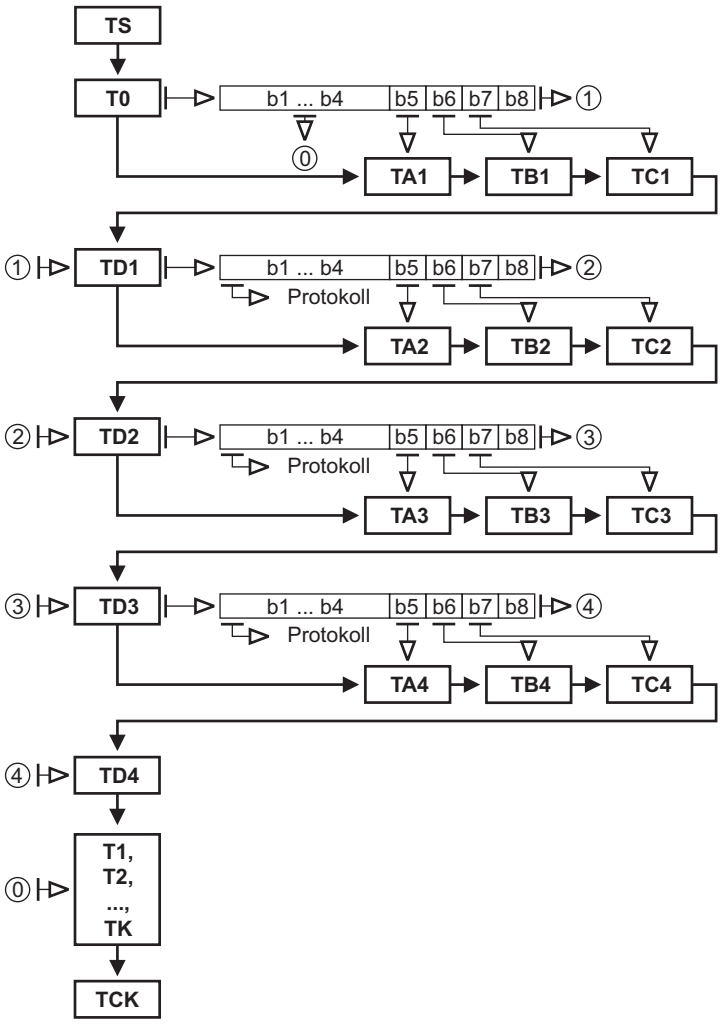

Bild 8.5 Der grundlegende Aufbau und die Datenelemente des ATR.

Der Datenstring und die Datenelemente des ATR sind in der ISO/IEC 7816-3 definiert und detailliert erläutert. Der grundlegende Aufbau eines ATR ist dabei wie folgt: Die ersten beiden Bytes, TS und T0 genannt, definieren verschiedene grundlegende Übertragungsparameter sowie das Vorhandensein von nachfolgenden Bytes. In den Interface Characters werden

[1] Siehe auch 13.10 „Atomare Abläufe“ auf Seite 522.

spezielle Parameter der Übertragungsprotokolle angegeben, die für die anschließende Datenübertragung von Bedeutung sind. Die Historical Characters beschreiben den grundlegenden Funktionsumfang der Chipkarte. Als letztes Byte des ATRs kann je nach Übertragungsprotokoll „The Check Character“ TCK gesendet werden, der eine Prüfsumme von vorangehenden ATR-Bytes ist.

8.1.1 Der Initial Character

Dieses mit TS bezeichnete Byte enthält die convention aller Daten des ATR sowie des nachfolgenden Kommunikationsablaufes. Zusätzlich beinhaltet TS ein charakteristisches Bitmuster, das vom Terminal zur Erkennung des Teilers benutzt werden kann. Dazu misst das Terminal die Zeit zwischen den ersten beiden fallenden Flanken in TS, teilt diese Zeit durch drei, und das Ergebnis ist die Zeitdauer für ein etu (*elementary time unit*). Da der für den ATR zu verwendende Teiler aber auf 372 festgelegt ist, findet in Terminals meist keine Auswertung dieses Synchronisationsmusters statt. Das erste Byte ist ein verpflichtender Bestandteil des ATR, es muss in jedem Fall gesendet werden. Für dieses Byte gibt es nur zwei erlaubte Kodierungen: '3B' in direct convention oder '3F' in inverse convention kodiert.

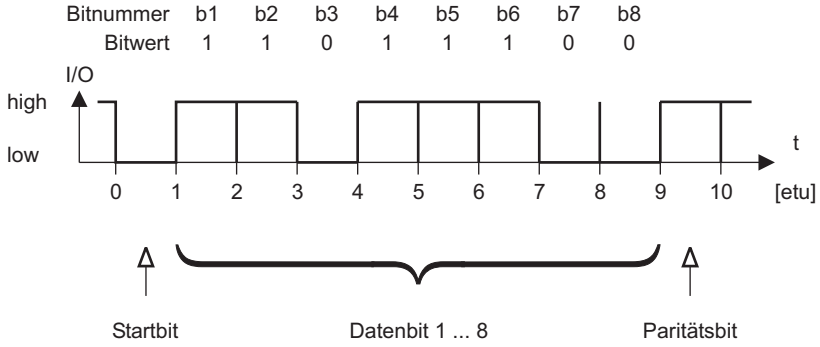

Bild 8.6 Zeitliche Darstellung des Initial Character TS bei direct convention. Dies wird durch den Wert '3B' = 0011 1011 angezeigt.

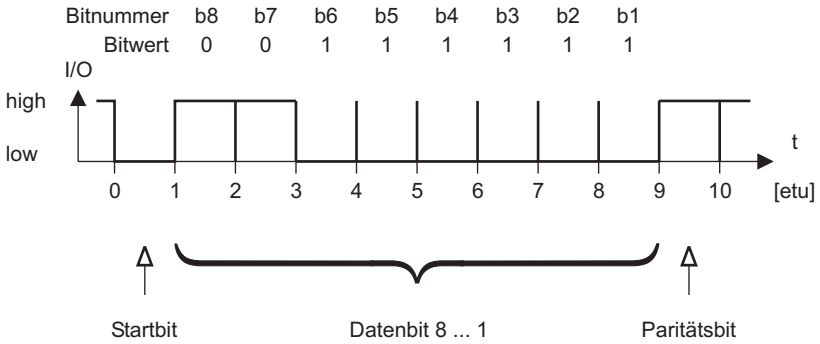

Bild 8.7 Zeitliche Darstellung des Initial Character TS bei inverse convention. Dies wird durch den Wert '3F' = 0011 1111 angezeigt.

Die in Deutschland übliche convention der Bytes ist direct. In Frankreich wird vornehmlich inverse convention benutzt. Die convention hat keinen Einfluss auf die Qualität der Übertragung. Natürlich gibt es bei den verschiedenen Betriebssystem-Herstellern einen in der Historie begründeten Vorzug, doch unterstützen alle Terminals und viele Chipkarten sowohl direct als auch inverse convention.

Tabelle 8.2 Die Kodierung des Initial Characters TS.

b8	b7	b6	b5	b4	b3	b2	b1	Bedeutung
				'3B'				direct convention
				'3F'				inverse convention

8.1.2 Der Format Character

Das zweite Byte, T0, dient zur Anzeige der nachfolgenden „Interface Characters“, wozu ein Bitfeld verwendet wird. Weiterhin enthält es noch die Anzahl der danach folgenden „Historical Characters“. Dieses Byte muss ebenfalls wie TS obligatorisch in jedem ATR enthalten sein.

Tabelle 8.3 Die Kodierung des Format Characters T0.

b8	b7	b6	b5	b4	b3	b2	b1	Bedeutung
…	…	…	…			X		Anzahl der Historical Characters [0 … 15]
…	…	…	1			…		TA_1 wird übertragen
…	…	1	…			…		TB_1 wird übertragen
…	1	…	…			…		TC_1 wird übertragen
1	…	…	…			…		TD_1 wird übertragen

8.1.3 Die Interface Characters

Die Interface Characters spezifizieren alle Übertragungsparameter der vorhandenen Übertragungsprotokolle. Sie setzen sich aus den Bytes TA_i, TB_i, TC_i und TD_i zusammen. Die Übertragung dieser Bytes im ATR ist jedoch optional und kann damit gegebenenfalls auch unterbleiben. Da für alle Übertragungsparameter der Übertragungsprotokolle voreingestellte Werte (*default values*) definiert sind, benötigt man für einen üblichen Kommunikationsablauf oft keine Interface Characters im ATR.

Die Interface Characters können in die Global Interface Characters und die Specific Interface Characters aufgeteilt werden. In den Global Interface Characters sind grundlegende Parameter für das Übertragungsprotokoll spezifiziert, wie z. B. der Teiler, der eine Bedeutung für alle folgenden Protokolle hat. Die Specific Interface Characters hingegen legen die Parameter für ein ganz bestimmtes Übertragungsprotokoll fest. Das typische Beispiel dafür ist die work waiting time für T=0.

Die Global Interface Characters gelten grundsätzlich für alle Protokolle. Aus historischen Gründen (am Anfang gab es nur T=0 in der ISO-Normung) sind einige dieser Global Inter-

face Characters nur für T=0 relevant. Falls T=0 nicht realisiert ist, kann man sie weglassen. Dann gelten die voreingestellten Werte.

Das Byte TD_i ist nur dazu da, die Verkettung der jeweils folgenden Interface Characters sicherzustellen. Dazu ist im höherwertigen Halbbyte von TD_i bitweise das Vorhandensein der nachfolgenden Interface Characters angezeigt. Dies ist analog der Kodierung des Format Characters T0. Das niederwertige Halbbyte von TD_i zeigt noch das jeweilige zur Verfügung stehende Übertragungsprotokoll an.

Ist das Byte TD_i nicht vorhanden, dann bedeutet dies, dass TA_{i+1}, TB_{i+1}, TC_{i+1} und TD_{i+1} nicht übertragen werden.

Tabelle 8.4 Die Kodierung eines TD_i-Bytes.

b8	b7	b6	b5	b4	b3	b2	b1	Bedeutung
…	…	…	…			X		Nummer des Übertragungsprotokolls [0 … 15]
…	…	…	1			…		TA_{i+1} wird übertragen
…	…	1	…			…		TB_{i+1} wird übertragen
…	1	…	…			…		TC_{i+1} wird übertragen
1	…	…	…			…		TD_{i+1} wird übertragen

Die nicht der Verknüpfung dienenden anderen Bytes der Interface Characters TA_i, TB_i und TC_i spezifizieren das oder die vorhandenen Übertragungsprotokolle. Sie haben nach ISO/IEC 7816-3 die im folgenden Text beschriebenen Bedeutungen.

8.1.3.1 Global Interface Character TA_1

Der Teiler (*clock rate conversion factor*) wird im höherwertigen Halbbyte kodiert. Im niederwertigen Halbbyte ist der Übertragungsanpassungsfaktor (*bit adjustment factor*) D kodiert.

Tabelle 8.5 Die Kodierung von TA_1.

b8	b7	b6	b5	b4	b3	b2	b1	Bedeutung
		X				…		FI
		…				X		DI

Aufgrund obiger Kodierungen des Teilers F und des Übertragungsanpassungsfaktors D können die typischen Übertragungsgeschwindigkeiten normgerecht festgelegt werden. Es gelten dabei die nachfolgenden Zusammenhänge. Die Zeitdauer eines Bits beim ATR und PPS, als „initial etu“ bezeichnet, ist auf folgenden Wert festgesetzt:

$$initial\ etu = \frac{372}{f} \cdot \mathrm{s}$$

In dem nach ATR und PPS folgenden Übertragungsprotokoll kann die Zeitdauer für ein Bit unabhängig vom ATR definiert werden. Diese Zeitdauer heißt „work etu“ und ist definiert als

$$work\ etu = \frac{F}{D} \cdot \frac{1}{f} \cdot \mathrm{s}$$

Somit ist es möglich, durch die beiden Parameter „bit rate adjustment factor D“ und den „clock rate conversion factor F“ die Übertragungsrate individuell zu verändern und anzupassen. Die angelegte Taktfrequenz ist als f abgekürzt und wird in den obigen Formeln in Hertz eingesetzt. Die Taktfrequenz fmax gibt die maximal erlaubte Taktfrequenz an. Der Standardwert für fmax ist 5 MHz.

Tabelle 8.6 Die Kodierung von Fi und fmax.

Bit 8 bis 5	0000	0001	0010	0011	0100	
Fi	372	372	558	744	1 116	
f_{max}	4 MHz	5 MHz	6 MHz	8 MHz	12 MHz	
Bit 8 bis 5	0101	0110	0111	1000	1001	
Fi	1 488	1 860	RFU	RFU	512	
f_{max}	16 MHz	20 MHz	—	—	5 MHz	
Bit 8 bis 5	1010	1011	1100	1101	1110	1111
Fi	768	1 024	1 536	2 048	RFU	RFU
f_{max}	7,5 MHz	10 MHz	15 MHz	20 MHz	—	—

Tabelle 8.7 Die Kodierung von Di.

Bit 4 bis 1	0000	0001	0010	0011	0100	0101	0110	0111
D	RFU	1	2	4	8	16	32	64
Bit 4 bis 1	1000	1001	1010	1011	1100	1101	1110	1111
D	12	20	RFU	RFU	RFU	RFU	RFU	RFU

8.1.3.2 Global Interface Character TA_i

Der Wert TA_i (= X || Y) ist gültig für $i > 2$ und T=15='F' in $TD_{(i-1)}$. TA_i enthält den Indikator X zum Anhalten des Takts (*clock stop indicator*). Dieser gibt an, auf welchem logischen Zustand sich die Taktleitung befinden muss, wenn der Takt angehalten wird. Zusätzlich enthält TA_i einen Indikator (*class indicator*) für die Klasse der Versorgungsspannung.

Tabelle 8.8 Die Kodierung von X.

Bit 8 und 7	00	01	10	11
X	nicht unterstützt	Zustand low	Zustand high	kein bevorzugter Zustand

Tabelle 8.9 Die Kodierung von Y. Alle nicht aufgeführten Werte sind für eine künftige Verwendung reserviert (RFU).

Bit 6 bis 1	00 0001	00 0010	00 0100
Y	Spannungsklasse A 4,5 V-5,5 V	Spannungsklasse B 2,7 V-3,3 V	Spannungsklasse C 1,62 V-1,98 V
Bit 6 bis 1	00 0011	00 0110	00 0111
Y	Spannungsklasse A und B	Spannungsklasse B und C	Spannungsklasse A, B und C

8.1.3.3 Global Interface Character TC_1

Hier wird eine zusätzliche Schutzzeit (*extra guardtime*) mit der Bezeichnung N als hexadezimaler vorzeichenloser und ganzzahliger Bytewert kodiert. Die zusätzliche Schutzzeit ist dabei definiert als zeitliche Verlängerung der Stoppbits. Der Wert N gibt die Anzahl der etu an, um die die Schutzzeit verlängert werden soll. TC_1 ist in seiner Bedeutung linear, mit der Ausnahme, dass N = 'FF' bedeutet, dass die Schutzzeit, die normalerweise zwei etu beträgt, bei T=1 auf 11 etu verkürzt werden muss. Bei T=0 bleibt die zusätzliche Schutzzeit auf N=12 etu, um die Fehlersignalisierung durch einen low-Pegel innerhalb der Schutzzeit weiterhin zu ermöglichen. In der Praxis bedeutet die bei T=1 auf 11 etu reduzierte Schutzzeit eine Geschwindigkeitssteigerung um circa 10 %, da ein Bit weniger gesendet werden muss.

Tabelle 8.10 Die Kodierung von TC_1.

b8	b7	b6	b5	b4	b3	b2	b1	Beschreibung
				X				zusätzliche Schutzzeit N mit dem Wertebereich 0 etu . . . 254 etu
1	1	1	1	1	1	1	1	X = 255 und T=0: zusätzliche Schutzzeit N = 12 etu X = 255 und T=1: zusätzliche Schutzzeit N = 11 etu

8.1.3.4 Specific Interface Character TC_2

Für das Übertragungsprotokoll T=0 ist der folgende Specific Interface Character definiert. Das Datenelement kodiert die waiting time WT mit dem positiven Ganzzahlwert WI und wird als TC_2 bezeichnet. Die waiting time ist die maximale Zeit zwischen den Startflanken zweier aufeinanderfolgender Bytes. Falls TC_2 im ATR nicht enthalten ist, beträgt der Standardwert für die waiting time integer WI = 10.

$$WT = WI \cdot 960 \cdot \frac{Fi}{f} \text{ work etu}$$

8.1.3.5 Specific Interface Character TA_i (i>2)

Das Byte TA_i enthält die maximale Länge des Informationsfeldes, das von der Karte empfangen werden kann (IFSC). Der Wert muss sich dabei zwischen 1 und 254 bewegen. Die beiden Grenzwerte dürfen ebenfalls verwendet werden. Die voreingestellte Größe von IFSC beträgt 32 Byte.

Tabelle 8.11 Die Kodierung von TA_i für i > 2.

b8	b7	b6	b5	b4	b3	b2	b1	Bedeutung
				X				IFSC

8.1.3.6 Specific Interface Character TB_i (i>2)

Das niederwertige Halbbyte beinhaltet in den vier Bits b4 bis b1 die Kodierung für die Zeichenwartezeit CWT[1], so dass gilt:

$$CWT = (11 + 2^{CWI}) \text{ work etu}$$

Das höherwertige Halbbyte enthält den Wert BWI, mit dem die Blockwartezeit BWT[2] folgendermaßen errechnet werden kann:

$$BWT = 11 \text{ work etu} + 2^{BWI} \cdot 960 \cdot \frac{Fd}{f}$$

Tabelle 8.12 Die Kodierung von TB_i für i>2.

b8	b7	b6	b5	b4	b3	b2	b1	Bedeutung
		…				X		CWI
		X				…		BWI

8.1.3.7 Specific Interface Character TC_i (i>2)

Das Bit b1 kodiert die Berechnungsweise des Fehlererkennungskodes. Mit dem Bit wird zwischen den beiden Möglichkeiten LRC und CRC unterschieden.

Tabelle 8.13 Die Kodierung von TC_i für i > 2.

b8	b7	b6	b5	b4	b3	b2	b1	Bedeutung
…	…	…	…	…	…	…	0	LRC wird benutzt
…	…	…	…	…	…	…	1	CRC wird benutzt
0	0	0	0	0	0	0	…	Reserviert für spätere Verwendung

[1] Siehe auch Abschnitt 9.3.2 „Übertragungsprotokoll T=1“ auf Seite 282.
[2] Siehe auch Abschnitt 9.3.2 „Übertragungsprotokoll T=1“ auf Seite 282.

8.1.3.8 Global Interface Character TA_2

Das Byte TA_2 zeigt die möglichen Modi an, die für den PPS verwendet werden können. Im Abschnitt 8.2 auf Seite 235 über den PPS ist dies genauer erläutert.

Tabelle 8.14 Die Kodierung von TA_2.

b8	b7	b6	b5	b4	b3	b2	b1	Bedeutung
0	…	…	…			…		Umschaltung zwischen negotiable mode und specific mode möglich.
1	…	…	…			…		Umschaltung zwischen negotiable mode und specific mode nicht möglich.
…	0	0	…			…		Reserviert für spätere Verwendung.
…	…	…	0			…		Übertragungsparameter sind explizit in den Interface Characters definiert.
…	…	…	1			…		Übertragungsparameter sind implizit in den Interface Characters definiert.
…	…	…	…			X		Protokoll T=X ist zu verwenden.

8.1.4 Die Historical Characters

Der Inhalt der Historical Characters war längere Zeit von keiner Norm festgelegt, was dazu führte, dass immer noch Festlegungen vorkommen, die nur für ein bestimmtes Betriebssystem gültig sind.

Viele Firmen geben in den zur Verfügung stehenden Bytes eine Bezeichnung für das Betriebssystem sowie eine dazugehörige Versionsnummer der ROM-Maske bzw. des Flash-OS an. Die Kodierung ist meist in ASCII, so dass sie einfach zu interpretieren ist. Die Existenz der Historical Characters innerhalb des ATRs ist nicht vorgeschrieben. Somit ist es möglich, sie komplett wegzulassen, was in manchen Fällen ein Vorteil ist, da der ATR kürzer wird und schneller ausgesendet werden kann.

Die internationale Norm ISO/IEC 7816-4 sieht zusätzlich zu den Historical Characters eine ATR-Datei vor. Diese Datei mit dem reservierten FID von '2F01' unter dem MF enthält weitere Informationen über den ATR. Sie ist als Erweiterung der in der Länge auf 15 Byte begrenzten Historical Characters gedacht. Der Inhalt der Datei mit nicht von der Norm festgelegter Struktur ist ASN.1-kodiert.

Die in der ATR-Datei oder in den Historical Characters enthaltenen Datenelemente können vielschichtige Informationen über die Chipkarte und das verwendete Betriebssystem sein. So können von der Karte unterstützte Funktionen für Dateiselektion, der impliziten Selektion und Informationen über den Mechanismus der logischen Kanäle dort angezeigt werden. Weiterhin können zusätzliche Daten über Kartenherausgeber, Seriennummer von Karte und Chip, Version der ROM-Maske, des Chips und des Betriebssystems dort untergebracht werden. Die Kodierung der entsprechenden Datenobjekte ist in den Normen ISO/IEC 7816-4 enthalten.

Nach ISO/IEC 7816-4 können die Historical Characters die drei folgenden Datenfelder enthalten: einen verpflichtenden Category Indicator, ein oder mehrere optionale Datenobjekte im Compact-TLV Format und einen optionalen Status Indicator.

Der Category Indicator wird in T1 übertragen. Er enthält Informationen über den Aufbau der Daten im ATR. Das Compact-TLV-Format enthält im ersten Nibble ein Kennzeichen (*tag*) und im zweiten Nibble die Länge der folgenden Daten. Die nach dem Category Indicator folgenden Daten umfassen Informationen über unterstützte Dienste und Fähigkeiten des Chipkarten-Betriebssystems, deren wichtigste in den folgenden Tabellen beschrieben sind. Der Status Indicator gibt den Lebenszustand der Chipkarte an.

Tabelle 8.15 Die Kodierung des Category Indicators in T1.

Wert	**Bedeutung**
'00'	Die Statusinformation befindet sich am Ende der Historical Characters.
'10'	Referenz auf eine DIR-Datei (*directory file*)
'80'	Falls vorhanden, befindet sich die Statusinformation in einem Compact-TLV-kodierten Datenobjekt.
'81' … '8F'	RFU
alle anderen Werte	Anwendungsspezifisch (*proprietary*)

Tabelle 8.16 Die Kodierung der anwendungsunabhängigen Kartendienste im Compact-TLV-kodierten Datenobjekt '31' (= Kennzeichen || Länge).

b8	**b7**	**b6**	**b5**	**b4**	**b3**	**b2**	**b1**	**Bedeutung**
1	…	…	…	…	…	…	…	Anwendungsselektion mit dem gesamten DF Name.
…	1	…	…	…	…	…	…	Anwendungsselektion mit einem partiellen DF Name.
…	…	1	…	…	…	…	…	Datenobjekte in der DIR-Datei verfügbar.
…	…	…	1	…	…	…	…	Datenobjekte in der ATR-Datei verfügbar.
…	…	…	…	1	…	…	…	Datenobjekte in der DIR-/ATR-Datei mittels Kommando READ BINARY lesbar.
…	…	…	…	0	…	…	…	Datenobjekte in der DIR-/ATRDatei mittels Kommando READ RECORD lesbar.
…	…	…	…	…	0	0	0	nicht benutzt

Tabelle 8.17 Die Kodierung der Kartenfähigkeiten (*first software function table*) in Compact-TLV kodierten-Datenobjekten. Das kombinierte Kennzeichen- und Längen-Byte kann dabei je nach Anzahl der Bytes für die Kartenfähigkeiten die Werte '71', '72' oder '73' annehmen.

b8	b7	b6	b5	b4	b3	b2	b1	Bedeutung
1	…	…	…	…	…	…	…	Selektion von DFs durch vollständigen DF Name.
…	1	…	…	…	…	…	…	Selektion von DFs durch partiellen DF Name.
…	…	1	…	…	…	…	…	Selektion von DFs durch Angabe des Pfades.
…	…	…	1	…	…	…	…	Selektion von DFs durch FID.
…	…	…	…	1	…	…	…	Implizite Selektion des DFs (mit der gewünschten Anwendung).
…	…	…	…	…	1	…	…	Short-FID werden unterstützt.
…	…	…	…	…	…	1	…	Record-Nummern werden unterstützt.
…	…	…	…	…	…	…	1	Record-Identifier werden unterstützt.

Tabelle 8.18 Die Kodierung der Kartenfähigkeiten (*second software function table*) in Compact-TLV kodierten-Datenobjekten. Das kombinierte Kennzeichen- und Längen-Byte kann dabei je nach Anzahl der Bytes für die Kartenfähigkeiten die Werte '71', '72' oder '73' annehmen.

b8	b7	b6	b5	b4	b3	b2	b1	Bedeutung
…	0	0	…	…	…	…	…	Das WRITE-Kommando verhält sich als einmaliges Schreiben (*one time write*).
…	0	1	…	…	…	…	…	Das WRITE-Kommando verhält sich anwendungsspezifisch (*proprietary*).
…	1	0	…	…	…	…	…	Das WRITE-Kommando verhält sich als logisches ODER.
…	1	1	…	…	…	…	…	Das WRITE-Kommando verhält sich als logisches UND.
…	…	…	…	1	…	…	…	Implizite Selektion des DFs (mit der gewünschten Anwendung).
…	…	…	…	…	x	x	x	Größe der Dateneinheiten in Nibbles und zur Potenz von 2 (001 = 2^2 = 2 Nibbles = 1 Byte).
0	…	…	0	0	…	…	…	nicht benutzt

Tabelle 8.19 Die Kodierung der Kartenfähigkeiten (*third software function table*) in Compact-TLV kodierten-Datenobjekten. Das kombinierte Kennzeichen- und Längen-Byte kann dabei je nach Anzahl der Bytes für die Kartenfähigkeiten die Werte '71', '72' oder '73' annehmen.

b8	b7	b6	b5	b4	b3	b2	b1	Bedeutung
1	…	…	…	…	…	…	…	Kommando-Chaining erlaubt.
…	1	…	…	…	…	…	…	Erweiterte L_c-/L_e-Felder.
…	…	…	1	0	…	…	…	Zuweisung der logischen Kanäle durch die Karte bei Kommando MANAGE CHANNEL.
…	…	…	0	1	…	…	…	Zuweisung der logischen Kanäle durch das Terminal bei Kommando MANAGE CHANNEL.
…	…	…	0	0	…	…	…	Logische Kanäle werden nicht unterstützt.
…	…	x	…	…	y	z	t	Maximale Zahl der logischen Kanäle (= 8x + 4y + 2z + t + 1).

8.1.5 Der Check Character

Dieses letzte Byte des ATRs ist eine XOR-Prüfsumme von Byte T0 bis zum letzten Byte vor TCK. Mit dieser Prüfsumme kann zusätzlich zur Paritätsprüfung die Richtigkeit der ATR-Übertragung festgestellt werden. Allerdings gibt es bei diesem Byte trotz des vermeintlich einfachen Aufbaus und der Berechnung einige signifikante Unterschiede zwischen den Übertragungsprotokollen.

Falls nur das Übertragungsprotokoll T=0 im ATR angezeigt ist, darf sich keine Prüfsumme TCK am Ende des ATRs befinden. Sie wird dann nicht gesendet, da bei T=0 die byteweise Fehlererkennung durch Paritätsprüfung und Wiederholung des fehlerhaften Bytes obligatorisch ist. Im Gegensatz dazu muss im Falle des Übertragungsprotokolls T=1 ein TCK-Byte im ATR vorhanden sein. Die Berechnung beginnt ab dem Byte T0 und endet beim letzten Interface Character bzw., sofern vorhanden, dem letzten Historical Character.

8.1.6 Praxisbeispiele für ATRs

Die folgenden Tabellen zeigen Beispiele aus der Praxis für unterschiedliche ATRs von Chipkarten und lassen sich sehr gut als Vorbild für Interpretationen von ATRs oder zur Definition von eigenen ATRs benutzen.

Tabelle 8.20 Beispiel für einen ATR einer Chipkarte mit dem Übertragungsprotokoll T=1 in direct convention.

Bezeichnung	Wert	Bedeutung	Bemerkung
TS	'3B'	direct convention	
T0	'B5'	Y1 = 1011 = 'B'	TA_1, TB_1, TD_1 folgen
		K = '5'	5 Historical Characters
TA_1	'11'	FI = 0001 = '1'	F = 372
		DI = 0001 = '1'	D = 1
TB_1	'00'	II = 0	I = 0
		PI1 = 0000 = '0'	Vpp-PIN ist nicht belegt
TD_1	'81'	Y2 = 1000 = '8'	TD_2 folgt
		T =1	
TD_2	'31'	Y2 = 0011 = '3'	TA_3 und TB_3 folgen
		T =1	
TA_3	'46'	I/O Pufferlänge = 70 Byte	Länge des ICC I/O Puffers (Schicht 7)
TB_3	'15'	BWI = '1'	Block Waiting Time
		CWI = '5'	Character Waiting Time
T1	'56'	"V"	"V 1.0"
T2	'20'	" "	
T3	'31'	"1"	
T4	'2E'	"."	
T5	'30'	"0"	
TCK	'1E'	The Check Character	XOR-Prüfsumme von T0 bis einschließlich T5

Tabelle 8.21 Beispiel für den ATR einer STARCOS-Chipkarte mit dem Übertragungsprotokoll T=1 in direct convention. Das Betriebssystem ist noch nicht komplettiert.

Bezeichnung	Wert	Bedeutung	Bemerkung
TS	'3B'	direct convention	
T0	'9C'	Y1 = '9' = 1001	TA_1 und TD_1 folgen
		K = 'C' = 12	12 Historical Characters
TA_1	'11'	FI = 0001 = '1'	F = 372
		DI = 0001 = '1'	D = 1
TD_1	'81'	'8' = 1000	TD2 folgt
		T=1	T=1 wird verwendet
TD_2	'21'	'2' = 0010	TB3 folgt
		T=1	T=1 wird verwendet
TB_3	'34'	BWI = 3	Block Waiting Time
		CWI = 4	Character Waiting Time
T1 . . . T12	'53' ‖ '43' ‖ '20' ‖ '53' ‖ '56' ‖ '20' ‖ '31' ‖ '2E' ‖ '31' ‖ '20'‖ '4E' ‖ '43'		"SC SV 1.1 NC"
TCK	'0F'	The Check Character	XOR-Prüfsumme von T0 bis einschließlich T12

Tabelle 8.22 Beispiel für den ATR einer STARCOS-Chipkarte mit dem Übertragungsprotokoll T=1 in direct convention. Das Betriebssystem ist komplettiert.

Bezeichnung	Wert	Bedeutung	Bemerkung
TS	'3B'	direct convention	
T0	'BF'	Y1 = 'F' = 1001	TA_1 und TD1 folgen
		K = 'B' = 11	11 Historical Characters
TA_1	'11'	FI = 0001 = '1'	F = 372
		DI = 0001 = '1'	D = 1
TD_1	'81'	'8' = 1000	TD_2 folgt
		T=1	T=1 wird verwendet
TD_2	'21'	'2' = 0010	TB_3 folgt
		T=1	T=1 wird verwendet
TB_3	'34'	CWI = 3	Character Waiting Time
		BWI = 4	Block Waiting Time
T1 ... T15	'53' ‖ '54' ‖ '41' ‖ '52' ‖ '43' ‖ '4F' ‖ '53' ‖ '32'‖ '31' ‖ '20' ‖ '43'		"STARCOS21 C"
TCK	'43'	The Check Character	XOR-Prüfsumme von T0 bis einschließlich T15

Tabelle 8.23 Beispiel für den ATR einer Chipkarte für GSM mit dem Übertragungsprotokoll T=0 in direct convention.

Bezeichnung	Wert	Bedeutung	Bemerkung
TS	'3B'	direct convention	
T0	'89'	Y1 = '8' = 1000	TD_1 folgt
		K = 9	9 Historical Characters
TD_1	'40'	'4' = 0100	TC_2 folgt
		T=0	T=0 wird verwendet
TC_2	'14'	WI = '14'	Work Waiting Time WWT = 20
T1 ... T9	'47' ‖ '47' ‖ '32' ‖ '34' ‖ '4D' ‖ '35' ‖ '32' ‖ '38' ‖ '30'		"GG24M5280"

Tabelle 8.24 Beispiel für den ATR einer Chipkarte für GSM mit dem Übertragungsprotokoll T=0 in inverse convention.

Bezeichnung	Wert	Bedeutung	Bemerkung
TS	'3F'	Inverse Convention	
T0	'2F'	Y1 = '2' = 0010 K = 'F' = 15	TB_1 folgt 15 Historical Characters
TB_1	'00'	PI1 = 00000 II = 00	Vpp ist nicht benutzt, die Programmierspannung für das EEPROM wird im Chip erzeugt.
T1 . . . T15	'80' ‖ '69' ‖ 'AE' ‖ '02' ‖ '02' ‖ '01' ‖ '36' ‖ '00' ‖ '00' ‖ '0A' ‖ '0E' ‖ '83' ‖ '3E' ‖ '9F' ‖ '16'		Spezifische Daten des Chipkartenherstellers

Tabelle 8.25 Beispiel für den ATR einer Chipkarte für Visa Cash mit dem Übertragungsprotokoll T=1 in direct convention.

Bezeichnung	Wert	Bedeutung	Bemerkung
TS	'3B'	direct convention	
T0	'E3'	Y1 = 'E' = 1110 K = 3	TB_1, TC_1 und TD_1 folgen 3 Historical Characters
TB_1	'00'	PI1 = 00000 II = 00	Vpp ist nicht benutzt, die Programmierspannung für das EEPROM wird im Chip erzeugt.
TC_1	'00'	N = 0	keine extra Guardtime
TD_1	'81'	'8' = 1000 T=1	TD_2 folgt T=1 wird verwendet
TD_2	'31'	'3' = 0011 T=1	TA_3 und TB_3 folgen T=1 wird verwendet
TA_3	'6F'	IFSC = '6F' = 111	Information Field Size der Chipkarte ist 111 Byte
TB_3	'45'	BWI = 4 CWI = 5	Block Waiting Time Character Waiting Time
T1	'80'	Category Indicator	Ein Datenobjekt im Compact-TLV Format folgt (nach ISO/IEC 7816-4)
T2	'31'	Card Service Data	Kennzeichen ('3') und Länge ('1') des Datenobjekts für die Card-Service-Daten
T3	'C0'	'C0' = 1100 0000	Anwendungsselektion mit DF Name als Ganzem oder in Teilen
TCK	'08'	The Check Character	XOR-Prüfsumme von T0 bis einschließlich T3

8.2 Protocol Parameter Selection (PPS)

In den Interface Characters des ATRs zeigt die Chipkarte verschiedene Parameter der Datenübertragung an, so z. B. das Übertragungsprotokoll und die Character Waiting Time. Möchte nun ein Terminal einen oder mehrere dieser Parameter ändern, so ist vor dem eigentlichen Ablauf des Übertragungsprotokolls eine Protocol Parameter Selection (PPS) nach ISO/IEC 7816-3 durchzuführen. Falls die Chipkarte dies erlaubt, kann das Terminal damit bestimmte Protokollparameter ändern. Die Protocol Parameter Selection (PPS) wurde vor 1997 Protocol Type Selection (PTS) genannt.

Die PPS kann in zwei unterschiedlichen Modi ausgeführt werden. Im negotiable mode werden die Standardwerte des Teilers F und des Übertragungsanpassungsfaktors D so lange beibehalten, bis ein PPS erfolgreich ausgeführt ist. Verwendet die Chipkarte den specific mode, dann sind die im ATR angegebenen Werte für F und D auch schon für die Datenübertragung des PPS obligatorisch. Welcher dieser beiden Modi von der Chipkarte unterstützt wird, zeigt letztere im TA_2 Byte an.

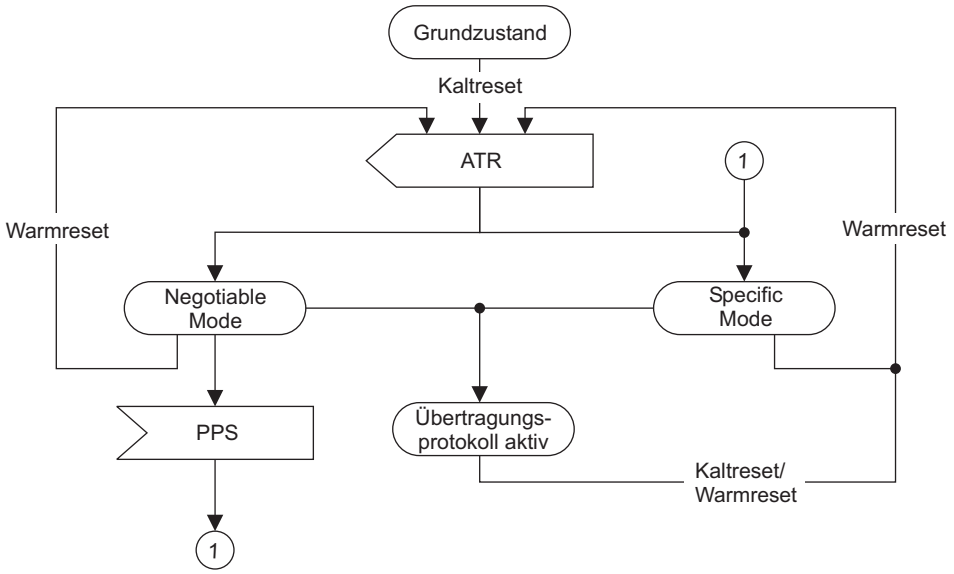

Bild 8.8 Zustandsdiagramm der beiden möglichen PPS-Modi nach ISO/IEC 7816-3: Negotiable Mode und Specific Mode.

Tabelle 8.26 Die Datenelemente des PPS und ihre Bezeichnungen nach ISO/IEC 7816-3.

Datenelement	Bezeichnung
PPSS	The Initial Character
PPS0	The Format Character
PPS_1, PPS_2, PPS_3	The Parameter Characters
PCK	The Check Character

Die PPS-Anfrage (PPS-Request) muss unmittelbar nach Empfang des ATR vom Terminal durchgeführt werden, wie dies auch Bild 8.10 auf der nächsten Seite zeigt. Erlaubt die Chipkarte die gewünschte Änderung der Protokollparameter, so sendet sie die empfangenen PPS-Bytes zum Terminal zurück. Dies ist im Prinzip eine Spiegelung der empfangenen Daten. Im anderen Fall sendet die Chipkarte nichts aus, und das Terminal muss erneut eine Reset-Sequenz fahren, damit die Chipkarte diesen Zustand verlässt. Der PPS kann nur ein einziges Mal direkt nach dem ATR ausgeführt werden, ein wiederholtes Absenden des PPS ist nach ISO/IEC 7816-3 nicht erlaubt.

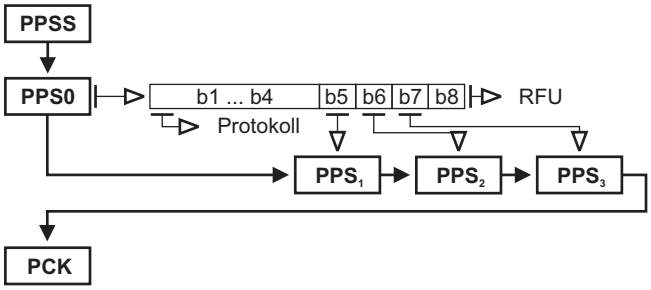

Bild 8.9 Der grundlegende Aufbau und die Datenelemente des PPS.

Tabelle 8.27 Die Kodierung von PPS0.

b8	b7	b6	b5	b4	b3	b2	b1	Bedeutung
…	…	…	…			X		Zu verwendendes Übertragungsprotokoll
…	…	…	1			…		PPS_1 ist vorhanden
…	…	1	…			…		PPS_2 ist vorhanden
…	1	…	…			…		PPS_3 ist vorhanden
0	…	…	…			…		Reserviert für zukünftige Anwendung

Tabelle 8.28 Die Kodierung von PPS_1.

b8	b7	b6	b5	b4	b3	b2	b1	Bedeutung
		X				…		F, kodiert analog TA_1
		…				X		D, kodiert analog TA_1

In der Praxis wird ein PPS immer dann verwendet, wenn die Übertragungszeit für den zeitlichen Ablauf der Anwendung einen deutlich sichtbaren Anteil hat. So ist ein PPS bei SIMs in GSM- und UMTS-Mobiltelefonen üblich, um den erheblichen Datentransfer zwischen Telefon und Chipkarte zu beschleunigen.

Das erste Byte „The Initial Character“ PPSS kennzeichnet für die Chipkarte eindeutig, dass vom Terminal direkt nach dem ATR eine PPS-Anfrage eingeleitet wird. Es hat deshalb immer den Wert 'FF' und ist obligatorisch in einem PPS zu senden.

Das dem PPSS folgende Datenelement „The Format Character“ PPS0 muss ebenfalls Bestandteil jedes PPS sein. Es können bis zu drei Bytes optional folgen, die „The Parameter Characters“ heißen und mit PPS_1, PPS_2 und PPS_3 bezeichnet werden. Sie kodieren diverse Parameter des nach dem PPS startenden Übertragungsprotokolls.

PPS_2 ermöglicht es dem Terminal, auf die SPU-Kontakte (*standard or proprietary use*) umzuschalten. Damit lässt sich beispielsweise das USB-Protokoll aktivieren. Das Datenelement PPS_3 ist für zukünftige Verwendung reserviert und deshalb hier noch nicht beschrieben.

Das letzte Byte im PPS heißt PCK und ist die XOR-Prüfsumme aller vorangegangenen Bytes ab PPSS. Es ist genau wie PPSS und PPS0 obligatorisch im PPS enthalten, im Gegensatz zu allen anderen Datenelementen, die optional sind.

Kann die Karte den PPS interpretieren und das Übertragungsprotokol entsprechend verändern, so sendet sie als Bestätigung den erhaltenen PPS zum Terminal zurück. Enthielt die PPS-Anfrage Anforderungen, die von der Chipkarte nicht ausgeführt werden können, dann wartet sie so lange, bis das Terminal einen Reset ausführt. Der größte Nachteil dieses Verfahrens ist der Zeitverlust bis zum Einsatz des eigentlichen Übertragungsprotokolls.

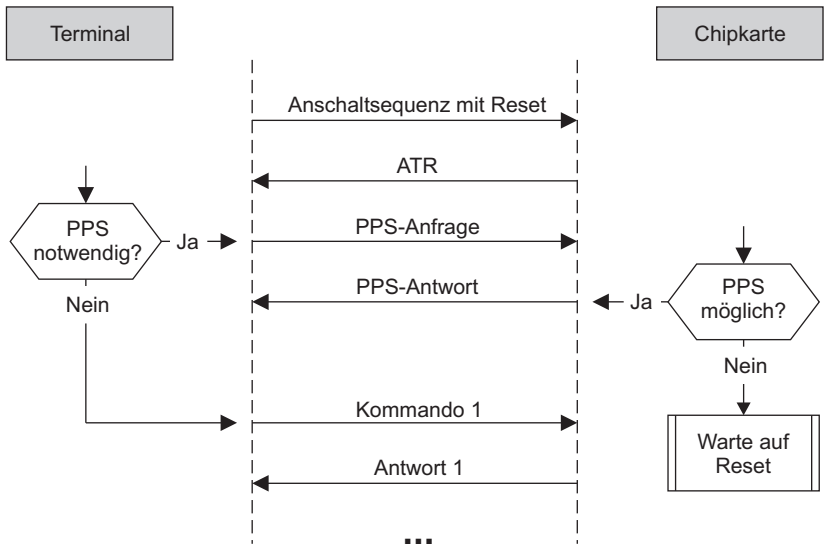

Bild 8.10 Ein typischer PPS-Ablauf einer SIM-Karte.

Der vorangehend beschriebene PPS würde nicht zur Protokollumschaltung bei Terminals funktionieren, die keinen PPS ausführen können, jedoch ein eigenes spezielles Übertragungsprotokoll haben. Genau dies ist aber z. B. bei den Kartentelefonen in Deutschland der Fall. Um dennoch die Möglichkeit eines Umschaltens des Übertragungsprotokolls zu haben, wurde dafür ein spezielles Verfahren erdacht.

Da alle Terminals bei einem unbekannten ATR mehrmals eine Resetsequenz ausführen, legte man fest, dass nach jedem Reset der Chipkarte diese das Übertragungsprotokoll wechselt. Ein Beispiel soll dies verdeutlichen: Beim ersten Reset sendet die Karte den ATR für T=14 und ist nach diesem auch mit T=14 kommunikationsbereit. Nach dem zweiten Reset sendet

sie einen ATR für T=1 und ist dann mit T=1 kommunikationsbereit, nach dem dritten Reset wieder mit T=14. Diese Lösung ist technisch gesehen nicht optimal, da sich ein Gerät nach einem Reset immer gleich verhalten sollte, doch stellt es eine durchaus praktikable Lösung für eine heterogene Terminalwelt dar.

Es ist möglich, diesen Nachteil zu relativieren, indem man nach einem Power-On-Reset der Chipkarte (Kaltreset) immer mit dem gleichen ATR antwortet. Ein Power-On-Reset wird von der Chipkarte immer direkt nach dem Stecken in ein Terminal und der Anschaltsequenz ausgeführt. Ein Reset über die Reset-Leitung (Warmreset) der Chipkarte führt hingegen zum Umschalten des Übertragungsprotokolls. Somit verhält sich die Chipkarte nach jedem „echten“ Einschalten gleich, und eine zusätzliche Auslösung des Resets führt zum Weiterschalten des Übertragungsprotokolls.

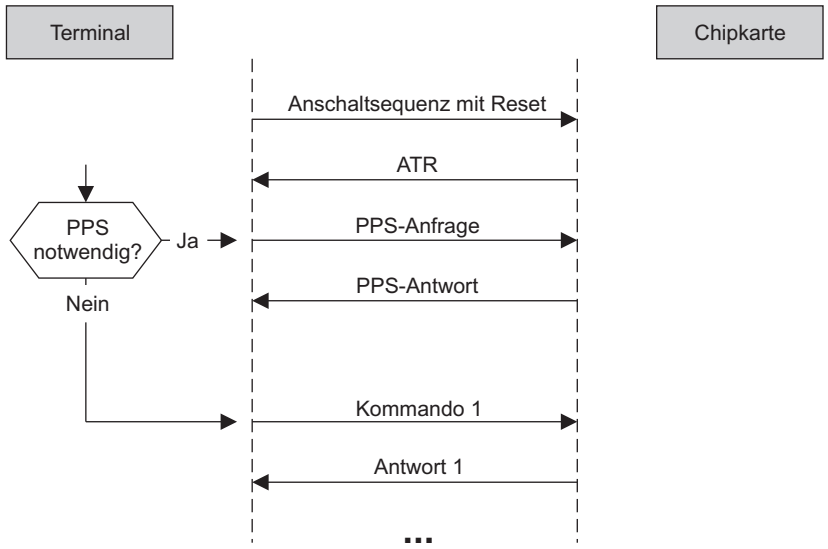

Bild 8.11 Ein möglicher Ablauf zur Umschaltung zwischen zwei durch die Chipkarte unterstützten Übertragungsprotokollen ohne PPS. Das Terminal muss bei diesem skizierten Ablauf keinen expliziten PPS ausführen, kann aber trotzdem mit einem Reset zwischen den beiden Protokollen umschalten.

8.3 Struktur der Nachrichten (APDUs)

Der gesamte Datenaustausch zwischen Chipkarte und Terminal findet unter der Verwendung von APDUs statt. Die Bezeichnung APDU ist die Abkürzung für den englischsprachigen Fachausdruck „application protocol data unit“. Er bezeichnet international genormte Dateneinheiten der Anwendungsschicht. Nach dem OSI-Schichtenmodell ist dies die Schicht 7. Bei Chipkarten ist diese Schicht direkt oberhalb der Übertragungsprotokolle angesiedelt. Die protokollabhängigen TPDUs (*transmission protocol data unit*) hingegen sind die Dateneinheiten der direkt darunter liegenden Schicht.

Unterschieden wird zwischen Kommando-APDUs (*command-APDU, C-APDU*), die Kommandos an die Chipkarte darstellen, und Antwort-APDUs (*response-APDU, R-APDU*), die

die Antworten der Chipkarte darauf sind. Vereinfacht ausgedrückt, sind APDUs eine Art Container, die ein vollständiges Kommando an die Chipkarte bzw. eine vollständige Antwort der Chipkarte enthalten. Sie werden vom Übertragungsprotokoll transparent, also ohne Veränderung oder Interpretation übertragen.

Die APDUs nach ISO/IEC 7816-4 sind dabei so aufgebaut, dass sie unabhängig vom Übertragungsprotokoll sind. Eine APDU muss also nicht in ihrem Inhalt oder Aufbau bei unterschiedlichen Übertragungsprotokollen geändert werden. Dies gilt natürlich vor allem für die beiden standardisierten Protokolle T=0 und T=1. Die Forderung nach Unabhängigkeit vom Übertragungsprotokoll beeinflusste den Aufbau der APDUs, da man sie sowohl mit dem Byteprotokoll T=0 als auch mit dem Blockprotokoll T=1 transparent übertragen muss.

8.3.1 Struktur der Kommando-APDUs

Eine Kommando-APDU setzt sich aus einem Header und einem Body zusammen. Der Body kann unterschiedliche Länge haben oder sogar ganz fehlen, wenn das dazugehörige Datenfeld leer ist.

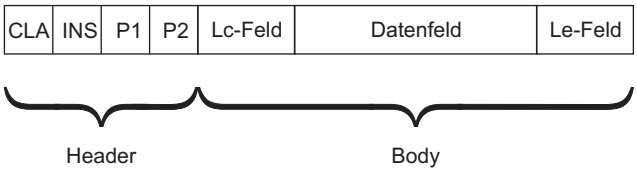

Bild 8.12 Die Struktur einer Kommando-APDU, bestehend aus Header und Body.

Der Header besteht aus den vier Elementen Class (CLA), Instruction (INS) und Parameter 1 (P1) und 2 (P2). Das Class-Byte wird auch dazu benutzt, um Anwendungen und ihren spezifischen Befehlssatz zu kennzeichnen. So verwendet GSM das Class-Byte 'A0', für firmenspezifische Kommandos (*private-use command*) wird meist die Kodierung '8X' benutzt. Kommandos nach ISO sind hingegen mit dem Class-Byte '0X' kodiert. Die Normung hat das Class-Byte zusätzlich zur Kennzeichnung für Secure Messaging und Logical Channels spezifiziert. Diese ist jedoch zu der erwähnten Nutzung als Anwendungskennzeichen kompatibel.

Das nächste Byte in der Kommando-APDU ist das Instruction-Byte, mit dem die eigentlichen Kommandos kodiert werden. Dabei kann fast der gesamte Adressraum dieses Bytes mit der Einschränkung nur geradzahliger Kodierungen ausgeschöpft werden. Dies war notwendig, weil das T=0-Protokoll ursprünglich die Programmierspannung durch ein um 1 erhöhtes Instruction-Byte im Procedure-Byte aktivieren konnte. Dieser Mechanismus wurde jedoch von der ISO/IEC-Normung ab circa 2004 nicht weiter unterstützt. Das Bit 1 wird statt dessen zur Anzeige von BER-TLV-kodierten Kommandos mit Kommando-Chaining (auch Schicht-7-Chaining genannt) benutzt. Mit diesen können auch Daten übertragen werden, die für ein einzelnes Kommando zu lang wären.

Die beiden Parameter-Bytes werden in erster Linie dazu verwendet, das durch das Instruction-Byte gewählte Kommando näher zu beschreiben. Sie dienen deshalb vor allem als Schalter, um verschiedene Optionen in dem Kommando anzuwählen. So nutzt man sie beispielswei-

se bei der Auswahl der unterschiedlichen Selektionsmöglichkeiten bei SELECT oder zur Angabe des Offsets bei READ BINARY.

Tabelle 8.29 Die wichtigsten Kodierungen des Class Byte CLA nach ISO/IEC 7816-4.

b8 . . . b5	b4	b3	b2	b1	Bedeutung
. . .	. . .	. . .	X	X	Nummer des Logical Channel
. . .	0	0	. . .	. . .	kein Secure Messaging
. . .	0	1	. . .	. . .	Secure Messaging nicht nach ISO, es wird ein eigenes Verfahren verwendet
. . .	1	0	. . .	. . .	Secure Messaging nach ISO, der Header ist nicht authentisch
. . .	1	1	. . .	. . .	Secure Messaging nach ISO, der Header ist authentisch
'0'	. . .	. . .	. . .	. . .	Struktur und Kodierung nach ISO/IEC 7816-4/-7/-8
'8', '9'	. . .	. . .	. . .	. . .	Struktur nach ISO/IEC 7816-4, anwendereigene Kodierung und Bedeutung der Kommandos und Antworten (private use)
'A'	. . .	. . .	. . .	. . .	Struktur und Kodierung nach ISO/IEC 7816-4, spezifiziert in zusätzlichen Dokumenten (z. B. TS 51.011)
'F'	1	1	1	1	Reserviert für PPS

Tabelle 8.30 Überblick über die Zuordnung von Class-Bytes zu bestimmten Anwendungen.

Class	Anwendung
'0X'	für die normierten Kommandos nach ISO/IEC 7816-4/-7/-8
'80'	für die elektronische Geldbörse nach EN 1546-3
'8X'	für anwendungs- und firmenspezifische Kommandos (*private use*)
'8X'	für Kreditkarten mit Chip nach EMV
'A0'	für das GSM-Mobilfunksystem TS 51.011

Der dem Header folgende Body, der bis auf eine Längenangabe weggelassen werden kann, erfüllt einen zweifachen Zweck: Zum Ersten legt er die Länge des Datenteils fest, der zur Karte gesendet wird (L_c-Feld – *length command*) und spezifiziert auch die Länge des von der Karte zurück zu sendenden Datenteils (L_e-Feld – *length expected*). Zum Zweiten enthält er die zum Kommando gehörenden Daten, die zur Karte gesendet werden. Falls das L_e-Feld den Wert '00' hat, erwartet das Terminal das Maximum der für dieses Kommando zur Verfügung stehenden Daten von der Karte. Dies ist die einzige Ausnahme in der numerischen Beschreibung der Längenangaben.

Üblicherweise haben L_e- und L_c-Feld immer eine Länge von 1 Byte. Es ist möglich, daraus ein L_e-/L_c-Feld mit je 3 Byte Länge zu machen. Damit könnten Längen bis 65 536 dargestellt werden, da das erste Byte als Escape-Sequenz '00' kodiert wäre. Die Norm definiert diese Längenangabe mit 3 Byte für zukünftige Anwendungen. Es gibt bereits erste Chipkarten-Betriebssysteme für Mikrocontroller mit größeren nichtflüchtigen Speichern, die Längenangaben mit 3 Byte unterstützen.

Durch die Kombination der vorgehend beschriebenen Teile der Kommando-APDU kann man die in Bild 8.14 aufgezeigten vier generellen Fälle, die so genannten cases, unterscheiden.

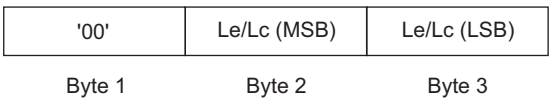

Bild 8.13 Die Struktur eines erweiterten L_c-/L_e-Feldes.

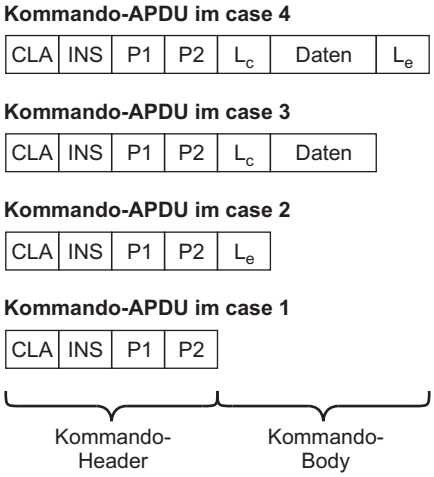

Bild 8.14 Die vier möglichen cases von Kommando-APDUs.

8.3.2 Struktur der Antwort-APDUs

Die von der Chipkarte auf eine Kommando-APDU gesendete Antwort-APDU besteht aus einem optionalen Body und einem obligatorisch zu sendenden Trailer und ist in Bild 8.15 gezeigt.

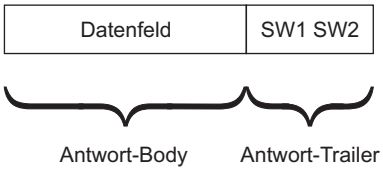

Bild 8.15 Die Struktur einer Antwort-APDU.

Der Body besteht aus dem Datenfeld, dessen Länge in der vorangehenden Kommando-APDU mit dem L_e-Byte festgelegt wurde. Diese Länge kann entgegen der Angabe im Le-Byte null sein, wenn die Chipkarte den Bearbeitungsprozess des Kommandos aufgrund eines Fehlers oder falschen Parameters abgebrochen hat. Dies wird dann in den beiden 1 Byte langen Status-Wörtern 1 und 2 im Trailer angezeigt.

Der Trailer muss von der Chipkarte in jedem Fall als Antwort auf ein Kommando gesendet werden. Die beiden Bytes SW1 und SW2, die auch als Returncode bezeichnet werden, beinhalten die Antwort auf das Kommando. So bedeutet beispielsweise '9000' als Returncode, dass ein Kommando erfolgreich und ohne Einschränkung ausgeführt wurde. Die Systematik, die den über 50 verschiedenen Kodes zugrunde liegt, ist in Bild 8.17 dargestellt.

Erhält man nach der Ausführung eines Kommandos den Returncode '63XX' oder '65XX', wurde der Zustand des nichtflüchtigen Speichers (d. h. EEPROM oder Flash) in der Chipkarte geändert. Empfängt man hingegen einen der restlichen Returncodes, die mit '6X' beginnen, wurde das Kommando abgebrochen, ohne dass im nichtflüchtigen Speicher eine Änderung eingetreten ist.

Anzumerken ist hier, dass es zwar eine Norm für die Returncodes gibt, doch bei manchen Anwendungen immer wieder davon abweichende Kodierungen benutzt werden. Die einzige Ausnahme stellt der Kode '9000' dar, der praktisch überall den Gutfall symbolisiert. Bei allen anderen Kodes sollte man immer in der zugrunde liegenden Spezifikation nachlesen, um die genaue Bedeutung zweifelsfrei herauszufinden.

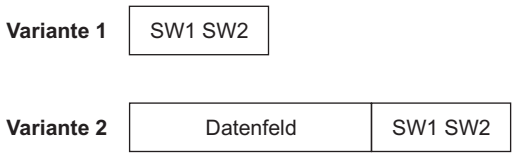

Bild 8.16 Die zwei Varianten der Antwort-APDUs.

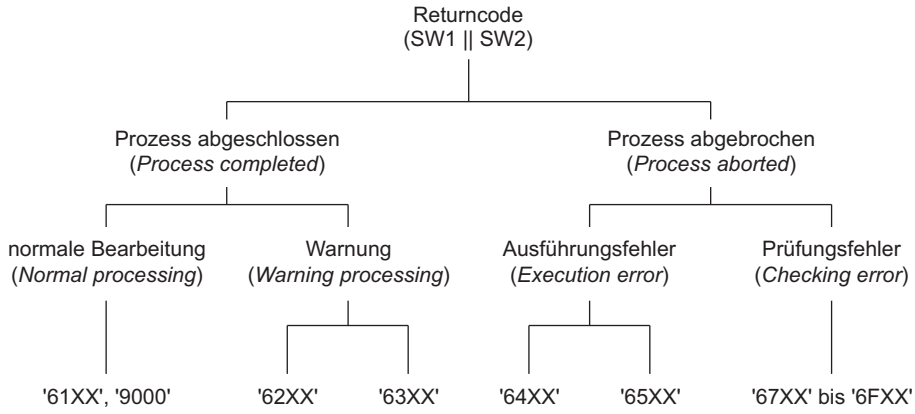

Bild 8.17 Die Systematik der Returncodes nach ISO/IEC 7816-4. Bei den Returncodes '63XX' und '65XX' wurden Daten im nichtflüchtigen Speicher (d. h. EEPROM oder Flash) verändert, bei den restlichen Returncodes mit '6X' nicht.

8.4 Sicherung der Datenübertragung

Der gesamte Datenaustausch zwischen Terminal und Chipkarte findet mittels digitaler elektrischer Impulse über die I/O-Leitung der Chipkarte statt. Es ist technisch einfach realisier-

bar, mit einem an der I/O-Kontaktfläche angelöteten Draht die gesamte Kommunikation aufzuzeichnen und später zu analysieren. Damit erhält man Kenntnis aller in beide Richtungen übertragenen Daten.

Etwas aufwändiger wäre es, die I/O-Kontaktfläche elektrisch zu isolieren und darauf einen Dummy-Kontakt aufzubringen. Zusätzlich wäre es notwendig, sowohl den Dummy-Kontakt als auch den Originalkontakt mit einer dünnen Leitung elektrisch mit einem Rechner zu verbinden. Mit dieser Anordnung ist es nun ein Leichtes, nur bestimmte Kommandos an die Chipkarte zuzulassen oder auch eigene Kommandos einzuspielen.

Diese beiden typischen Angriffe würden nur dann zum Erfolg führen, wenn geheime Daten ungesichert über die I/O-Leitung gingen. Die Übertragung der Daten sollte deshalb grundsätzlich so ausgelegt sein, dass ein Angreifer immer die Datenübertragung mithören und auch eigene Übertragungsblöcke in das Protokoll einfügen kann, ohne einen Vorteil aus einem dieser Angriffe ziehen zu können.

Um nun solche oder auch noch aufwändigere Angriffe abzuwehren, gibt es verschiedene Mechanismen und Verfahren. Diese bezeichnet man allgemein als Secure Messaging (SM). Sie sind nicht chipkartenspezifisch, da sie schon lange im Bereich der Datenfernübertragung verwendet werden. Das Spezifische im Chipkartenbereich ist, dass weder die Rechenleistung der beiden Kommunikationsteilnehmer noch die Übertragungsgeschwindigkeit allzu hoch sind. Deshalb wurden die allgemein verwendeten Standardverfahren auf die Vorgaben der Chipkartenwelt abgespeckt, wobei die Sicherheit der Verfahren keinerlei Einbußen erleiden durfte.

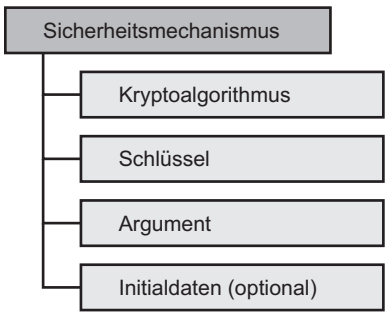

Bild 8.18 Notwendige Daten und Funktionen für einen Mechanismus zur gesicherten Datenübertragung.

Der Zweck der gesicherten Datenübertragung ist also, die Authentizität und bei Bedarf die Vertraulichkeit der übertragenen Daten oder Teile davon sicherzustellen. Um diese Forderung zu erfüllen, werden verschiedene Sicherheitsmechanismen benutzt. Ein Sicherheitsmechanismus ist definiert als eine Funktion, die Folgendes benötigt: einen kryptografischen Algorithmus, einen Schlüssel, ein Argument und bei Bedarf Initialdaten. Eine Rahmenbedingung muss zusätzlich erfüllt sein: Alle Sicherheitsmechanismen müssen sich gegenüber den vorhandenen Protokollschichten transparent verhalten, so dass bereits genormte und vorhandene Verfahren von einer gesicherten Datenübertragung nicht beeinträchtigt werden. Dies betrifft insbesondere die beiden Übertragungsprotokolle T=0 und T=1 sowie die allgemein verwendeten und genormten Kommandos für Chipkarten.

Vor der Ausführung dieser Verfahren müssen sich beide Kommunikationspartner über den verwendeten kryptografischen Algorithmus und einen gemeinsamen geheimen Schlüssel dafür einigen. Auf diesem Schlüssel basiert, ganz nach Kerckhoffs, die gesamte Sicherheit des Verfahrens. Würde dieser Schlüssel bekannt, wäre das Secure Messaging nur mehr eine die Datenübertragungsgeschwindigkeit herabsetzende, allgemein bekannte und zusätzliche Prüfsumme, mit der man höchstens Übertragungsfehler aufdecken kann.

Seit Jahren existieren verschiedene Variationen von Verfahren für Secure Messaging. Anfangs waren sie relativ starr und für eine bestimmte Anwendung maßgeschneidert. Vom Standpunkt der Sicherheit aus gab es meistens keinen Einwand. Allerdings setzte sich keines dieser Verfahren international so weit durch oder war so flexibel, dass es international normiert wurde.

Die Forderungen nach Transparenz zu vorhandenen Kommandos, zwei verschiedenen Übertragungsprotokollen und einer möglichst hohen Variabilität hatten zur Folge, dass ein sehr flexibles, aber dadurch auch aufwändiges und komplexes Verfahren für Secure Messaging in der ISO/IEC 7816-4 und erweiterte Funktionen dazu in der ISO/IEC 7816-8 genormt wurden.

Allerdings haben sich daneben weitere Verfahren zum kryptografischen Schutz der Datenübertragung etabliert. Im Bereich der Telekommunikation wird ein sehr mächtiges und auch umfangreiches Verfahren in der TS 43.048 festgelegt. Dies soll vor allem eine Ende-zu-Ende-Kommunikation zwischen (U)SIM und Hintergrundsystem schützen. Bei Global Platform sind ebenfalls geschützte Nachrichtenübertragungen spezifiziert, die ein analoges Schutzniveau wie die bereits genannten bieten. Diese haben die Bezeichnungen Secure Channel Protocol 01 respektive 02. Die CEN-Norm 14890 legt ebenfalls ein Secure Messaging-Verfahren fest, das vor allem für den Einsatz im Umfeld von digitalen Signaturen vorgesehen ist. Sofern TCP/IP als logische Kommunikationsschicht benutzt wird, nutzt man konsequenterweise SSL (*secure socket layer*) oder TLS (*transport layer security*) zur Sicherung der Datenübertragung.

Die Grundlage des Secure Messaging nach ISO/IEC 7816-4 stellt dabei die Einbettung aller Nutzdaten in TLV-kodierte Datenobjekte dar. Drei verschiedene Typen von Datenobjekten sind definiert:

- Datenobjekte für Klartext enthalten Daten im Klartext (z. B. Datenteil einer APDU).
- Datenobjekte für Sicherheitsmechanismen enthalten die Ergebnisse von Sicherheitsmechanismen (z. B. MAC).
- Datenobjekte für Hilfsfunktionen enthalten Kontrollinformationen für Secure Messaging (z. B. verwendete Paddingmethode).

Im Class-Byte wird angezeigt, ob Secure Messaging für das Kommando benutzt wird. Dabei kann in den beiden zur Verfügung stehenden Bits kodiert werden, ob das Verfahren nach ISO/IEC 7816-4/8 verwendet wird und ob der Header in die kryptografische Prüfsumme (CCS) mit eingeschlossen ist.[1] Ist der Header bei der Berechnung mit eingeschlossen, so ist er authentisch, da er während der Übertragung nicht unbemerkt verändert werden kann.

[1] Zur Kodierung des Class-Byte siehe Abschnitt 8.3.1 „Struktur der Kommando-APDUs“ auf Seite 239.

8.4.1 Datenobjekte für Klartext

Nach der Norm müssen alle Daten, die nicht BER-TLV-kodiert sind, in Datenobjekte eingebettet, d. h. gekapselt werden. Dazu gibt es verschiedene Kennzeichen, die in der Tabelle 8.31 aufgeführt sind. Das Bit 1 des Kennzeichens legt fest, ob ein Datenobjekt in die Berechnung der kryptografischen Prüfsumme mit einbezogen wird. So bedeutet ein nicht gesetztes Bit (z. B. 'B0'), dass das Datenobjekt nicht in die Berechnung mit einfließt und ein gesetztes Bit (z. B. 'B1'), dass es mit einfließt.

Tabelle 8.31 Die Kennzeichen für Klartext-Datenobjekte.

Kennzeichen	Bedeutung
'B0', 'B1'	BER-TLV-kodiert und enthält Datenobjekte, die im Zusammenhang mit Secure Messaging stehen
'B2', 'B3'	BER-TLV-kodiert und enthält Datenobjekte, die in keinem Zusammenhang mit Secure Messaging stehen
'80', '81'	keine BER-TLV-kodierten Daten
'99'	Zustandsinformationen über Secure Messaging

8.4.2 Datenobjekte für Sicherheitsmechanismen

Die Datenobjekte für Sicherheitsmechanismen teilen sich in Datenobjekte für Authentizität und in die für Vertraulichkeit auf. Die dafür definierten Kennzeichen befinden sich in den Tabellen 8.32 „Datenobjekte für Sicherheitsmechanismen“ auf Seite 245 und 8.33 „Datenobjekte für Sicherheitsmechanismen“ auf Seite 245.

Tabelle 8.32 Die Kennzeichen für Datenobjekte für Vertraulichkeit.

Kennzeichen	Bedeutung
'82', '83'	Kennzeichen für ein Kryptogramm: der Klartext ist BER-TLV-kodiert und schließt Datenobjekte für Secure Messaging ein.
'84', '85'	Kennzeichen für ein Kryptogramm: der Klartext ist BER-TLV-kodiert und schließt keine Datenobjekte für Secure Messaging ein.
'86', '87'	Kennzeichen für die verwendete Padding-Methode 01: Padding mit '80 00 . . . ' 02: ohne Padding

Tabelle 8.33 Die Kennzeichen für Datenobjekte für Authentizität.

Kennzeichen	Bedeutung
'8E'	kryptografische Prüfsumme
'9A', 'BA'	Eingangswert für eine digitale Signatur
'9E'	digitale Signatur

Der Oberbegriff Authentizität bezeichnet dabei alle Datenobjekte, die mit kryptografischen Prüfsummen und digitalen Signaturen zu tun haben. Die Verschlüsselung von Daten und

die dafür notwendige Kennzeichnung während Secure Messaging fällt unter den Begriff der Vertraulichkeit. Je nach verwendetem Verfahren muss man aus den beiden Tabellen die zutreffenden Kennzeichen suchen und bei der geschützten Datenübertragung benutzen.

8.4.3 Datenobjekte für Hilfsfunktionen

Für die Abstimmung der Rahmenbedingungen bei Secure Messaging benutzt man die Datenobjekte für Hilfsfunktionen. Mit diesen Datenobjekten tauschen die beiden Kommunikationspartner Informationen über den verwendeten kryptografischen Algorithmus, benutzte Schlüssel, Initialdaten und ähnliche grundlegende Informationen aus. Dies könnte theoretisch bei jeder übertragenen APDU bzw. selbst bei Kommando und Antwort unterschiedlich sein. In der Praxis werden diese Datenobjekte jedoch selten benutzt, da alle Rahmenbedingungen für Secure Messaging implizit definiert sind und nicht erst im Laufe der Kommunikation festgelegt werden müssen.

Auf der Grundlage der oben nur überblickshaft aufgezeigten Möglichkeiten, die das Secure Messaging nach ISO/IEC 7816-4/8 bietet, sind im Folgendem zwei grundlegende Verfahren dargestellt. Diese sind möglichst einfach gehalten, um das Verständnis der komplizierten Mechanismen zu erleichtern. Es gibt aufgrund der sehr hohen Flexibilität noch viele weitere und auch aufwändigere Kombinationsmöglichkeiten der Sicherheitsmechanismen. Die beiden hier gezeigten Verfahren stellen eine Synthese aus Einfachheit und möglicher Sicherheit dar.

Das vorgestellte Authentic-Verfahren schützt die Anwendungsdaten, d. h. die APDU, mit einer kryptografischen Prüfsumme (CCS, MAC) gegen Manipulationen während der Übertragung. Das Combined-Verfahren wird hingegen benutzt, um die Anwendungsdaten komplett zu verschlüsseln, so dass ein Angreifer keinerlei Rückschlüsse auf die Dateninhalte der übertragenen Kommandos und die erhaltenen Antworten ziehen kann. Nur in Verbindung mit einem dieser beiden Verfahren ist die Verwendung eines Sendefolgezählers zu sehen. Dieser Zähler, dessen Startwert eine Zufallszahl ist, wird mit jedem Kommando und jeder Antwort inkrementiert. Dadurch können die beiden Kommunikationspartner feststellen, ob ein Kommando oder eine Antwort verloren ging oder eingestreut wurde. In Verbindung mit dem Combined-Verfahren ermöglicht ein Sendefolgezähler auch, dass gleiche APDUs verschiedenes Aussehen haben („diversity“).

8.4.4 Das Authentic-Verfahren

Das Authentic-Verfahren gewährleistet eine authentische, also nicht fälschbare Übertragung von APDUs. Der Empfänger einer APDU, d. h. eines Kommandos oder einer Antwort, kann feststellen, ob diese während der Übertragung verändert wurde. Damit ist es einem Angreifer nicht mehr möglich, innerhalb einer APDU Daten zu ändern, ohne dass dies der Empfänger bemerkt. Bild 8.19 auf der nächsten Seite zeigt dazu den prinzipiellen Ablauf der Erstellung einer Kommando-APDU.

Die Benutzung des Verfahrens wird dabei durch ein Bit im Class-Byte angezeigt, so dass sich der Empfänger entsprechend verhalten und die erhaltene APDU auf Authentizität prüfen kann. Die APDUs selber werden im Klartext übertragen und sind nicht verschlüsselt.

Die übertragenen Daten sind also nach wie vor öffentlich und können bei geeigneter Manipulation der Übertragungsstrecke vom Angreifer empfangen und ausgewertet werden. Dies muss nicht unbedingt von Nachteil sein, da es auch aus datenschutzrechtlicher Sicht besser ist, nicht geheime Daten auch öffentlich zu übertragen. Damit ist dem Benutzer zumindest theoretisch die Möglichkeit gegeben, zu verfolgen, welche Daten von seiner Chipkarte und dem Terminal ausgetauscht werden.

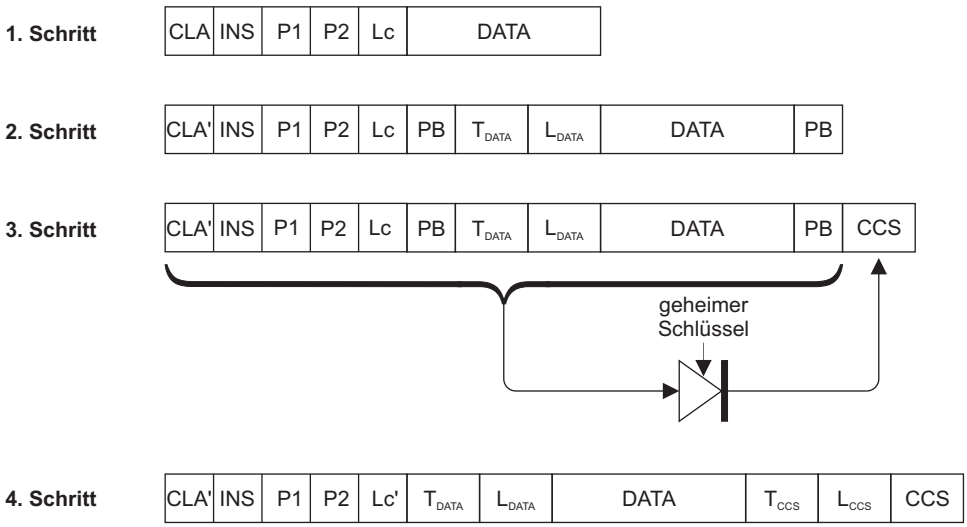

Bild 8.19 Erstellung einer Kommando-APDU im Authentic-Verfahren. Es wird ein Case-3-Kommando (z. B. UPDATE BINARY) verwendet, dessen Header in die kryptografische Prüfsumme (CCS) mit einbezogen ist. Die Erstellung einer Antwort-APDU verläuft analog. Die Padding Bytes sind als PB abgekürzt. Schritt 1 zeigt das Ausgangsformat der APDU, das in Schritt 2 in TLV-kodierte Daten mit Padding der Datenobjekte auf ein Vielfaches von 8 transformiert wird. Im Schritt 3 wird die kryptografische Prüfsumme (CCS) berechnet und dann im 4. Schritt die APDU mit einem TLV-kodierten Datenobjekt, das die CCS enthält, ergänzt.

Zur Berechnung der kryptografischen Prüfsumme kann im Prinzip jeder Blockverschlüsselungsalgorithmus verwendet werden. In den folgenden Ausführungen wurde dabei aus praktischen Gründen immer vom AES ausgegangen, der eine feste Blocklänge von 8 Byte besitzt. Deshalb muss man die einzelnen Datenobjekte auf ein Vielfaches der Länge von 8 Byte auffüllen, was man padding nennt. Die Datenobjekte, die bereits ein Vielfaches der Länge von 8 Byte haben, werden dabei trotzdem mit einem weiteren Block verlängert. Nach dem Auffüllen wird mit dem AES-Algorithmus im CBC-Mode eine kryptografische Prüfsumme (CCS) über die gesamte APDU errechnet. Diese 8 Byte lange Prüfsumme wird unter Weglassen der vier niederwertigen Bytes als TLV-kodiertes Datenobjekt direkt an die APDU angehängt, wobei nach der Berechnung der kryptografischen Prüfsumme alle Paddingbytes entfernt werden. Die mit den vorhergehenden Verfahren umgestaltete APDU wird dann über die Schnittstelle gesendet. Somit verlängert sich eine APDU bei diesem Authentic-Verfahren um 8 Byte, was die Übertragungsgeschwindigkeit bei den üblichen Übertragungsblockgrößen nur wenig absenkt.

In den Datenobjekten für die Kontrollstrukturen kann unter anderem explizit angezeigt werden, welcher Algorithmus und welche Paddingmethode Verwendung finden. Hier wird aus Vereinfachungsgründen angenommen, dass Chipkarte und Terminal implizit alle Parameter des verwendeten Secure Messaging kennen. Trifft diese geschützte APDU nun beim Empfänger ein, verlängert sie dieser wieder auf ein Vielfaches von 8 Byte und berechnet seinerseits einen MAC über die APDU. Durch Vergleich des empfangenen, vom Sender berechneten MACs mit dem selber berechneten kann nun der Empfänger feststellen, ob die APDU während der Datenübertragung verändert wurde.

Voraussetzung zur Berechnung einer kryptografischen Prüfsumme ist ein geheimer AES-Schlüssel, der beiden Kommunikationspartnern bekannt sein muss. Dieser Schlüssel muss deshalb geheim sein, weil sonst ein Angreifer die Möglichkeit hätte, die authentische Kommunikation zu brechen. Er könnte eine APDU abfangen, diese nach seinen Wünschen verändern und dann einen neuen korrekten MAC berechnen. Nachdem er diesen neuen MAC berechnet hat, muss er ihn nur noch gegen den ursprünglichen austauschen und die so neu erzeugte APDU wieder einspielen.

Um den Schlüssel zur MAC-Bildung gegen Angriffe mit bekannten Klartext-Schlüsseltext-Paaren besser zu schützen, verwendet man üblicherweise dynamische Schlüssel. Diese werden durch Verschlüsselung einer zwischen Terminal und Chipkarte vorab ausgetauschten Zufallszahl mit einem beiden Kommunikationsteilnehmern bekannten gemeinsamen Schlüssel erzeugt.[1]

Die zusätzlichen Schritte zur Übertragung und zum Empfang einer mit dem Authentic-Verfahren geschützten APDU reduzieren natürlich die effektive Übertragungsgeschwindigkeit. Im Durchschnitt kann in etwa eine Halbierung der Übertragungsgeschwindigkeit gegenüber Klartext angenommen werden.

8.4.5 Das Combined-Verfahren

Das Combined-Verfahren stellt gegenüber dem Authentic-Verfahren die nächste Stufe an Sicherheit dar. Die Übertragung des Datenteils der APDU findet hier nicht mehr im Klartext statt, sondern verschlüsselt. Dieses Verfahren setzt auf dem Authentic-Verfahren auf und ist in Bild 8.20 auf der nächsten Seite skizziert.

Beim Combined-Verfahren werden die analog dem Authentic-Verfahren mit einer kryptografischen Prüfsumme gesicherten Datenobjekte wiederum auf ein Vielfaches von 8 Byte vergrößert (*padding*) und mit dem AES im CBC-Mode verschlüsselt. Ausgespart bleibt dabei der Header, was aus Kompatibilitätsgründen zum T=0-Protokoll notwendig ist. Möchte man den Header ebenfalls verschlüsseln, so dass nicht mehr erkennbar ist, welches Kommando zur Karte gesendet wird, ist bei T=0 das Kommando ENVELOPE notwendig. Im Class-Byte dient ein Bit als Indikator für Secure Messaging. Nach der Verschlüsselung erfolgt die Übertragung über die Schnittstelle. Da der Empfänger den geheimen Schlüssel kennt, mit dessen Hilfe die Verschlüsselungsoperation stattgefunden hat, kann er die APDU damit wieder entschlüsseln. Anschließend wird geprüft, ob die Entschlüsselung korrekt war, indem man auf der gleichen Ebene der Übertragungsschicht die angehängte kryptografische Prüfsumme nachrechnet.

[1] Siehe dazu auch 7.7.4 „Dynamische Schlüssel“ auf Seite 196.

Ein an der I/O-Leitung horchender Angreifer kann bei Anwendung dieses Verfahrens nicht erfahren, welche Daten im Kommando und der Antwort zwischen Karte und Terminal ausgetauscht werden. Auch besteht keine Möglichkeit, innerhalb der APDU einen der verschlüsselten Datenblöcke auszutauschen, da diese durch den im CBC-Mode betriebenen AES miteinander verknüpft sind. Damit würde vom Empfänger ein Austausch sofort bemerkt.

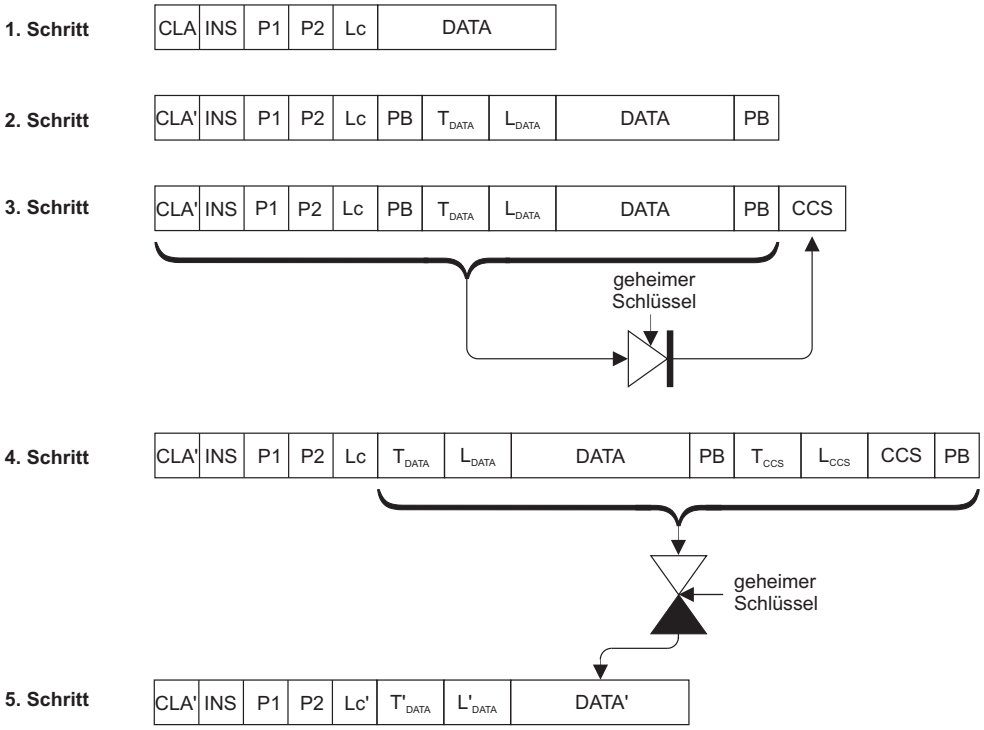

Bild 8.20 Erstellung einer Kommando-APDU im Combined-Verfahren. Es wird ein Case-3-Kommando (z. B. UPDATE BINARY) verwendet, dessen Header in die kryptografische Prüfsumme (CCS) mit einbezogen ist. Die Erstellung einer Antwort-APDU läuft analog. Die Padding Bytes sind als PB abgekürzt. Schritt 1 zeigt das Ausgangsformat der APDU, das in Schritt 2 in TLV-kodierte Daten mit Padding der Datenobjekte auf ein Vielfaches von 8 transformiert wird. Im Schritt 3 wird die kryptografische Prüfsumme (CCS) berechnet und dann im 4. Schritt der Datenteil der APDU verschlüsselt, so dass die veränderte APDU im 5. Schritt übertragen werden kann.

Zum Thema des kryptografischen Algorithmus gilt hier das Gleiche, was bereits beim Authentic-Verfahren angemerkt wurde. Prinzipiell kann jeder Blockverschlüsselungsalgorithmus verwendet werden. Die Schlüssel sollten ebenso wie im Authentic-Verfahren nur temporärer Natur sein, so dass für jede Sitzung ein eigens abgeleiteter Schlüssel Verwendung findet.[1] Betrachtet man nun die sicherheitstechnischen Vorteile, so wäre eine generelle Benutzung des Combined-Verfahrens für alle APDUs empfehlenswert. Doch mit der Erhöhung der Sicherheit ist auch eine Senkung der Datenübertragungsgeschwindigkeit verbunden.

[1] Siehe auch 7.7.1 „Abgeleitete Schlüssel“ auf Seite 195.

Die Einbuße an Übertragungsgeschwindigkeit ist erheblich. Der Unterschied zwischen der ungeschützten und der im Combined-Verfahren übertragenen APDU kann in guter Näherung mit dem Zeitfaktor 4 angenommen werden. Der zeitliche Unterschied zwischen Authentic- und Combined-Verfahren beläuft sich damit immerhin noch auf den Faktor 2. Es ist deshalb in jedem Einzelfall sehr genau zu prüfen, welche Daten auf solch aufwändige, aber sichere Weise übertragen werden sollen.

8.4.6 Sendefolgezähler (send sequence counter)

Gesicherte Datenübertragung mit dem Mechanismus eines Sendefolgezählers (*send sequence counter*) ist kein eigenständiges Sicherungsverfahren. Es kann nur sinnvoll in Verbindung mit dem Authentic- oder Combined-Verfahren eingesetzt werden, da sonst der Sendefolgezähler durch einen Angreifer unbemerkt verändert werden könnte.

Das Prinzip besteht darin, dass jede APDU abhängig von ihrem Sendezeitpunkt eine Folgenummer erhält. Damit würde das Entfernen oder Einfügen einer APDU im Protokollverlauf sofort auffallen und entsprechende Maßnahmen wie den Abbruch der Kommunikation seitens des Empfängers zur Folge haben.

Die Funktion basiert auf einem Zähler, dessen Startwert eine Zufallszahl ist und die am Anfang des Kommunikationsablaufes von der Chipkarte auf Anforderung an das Terminal gesendet wird. Bei jeder Übertragung einer APDU wird der Zähler inkrementiert. Die Länge des Zählers sollte nicht zu kurz sein, wegen des zusätzlichen Übertragungsbedarfs aber auch nicht zu lang. Für die folgenden Betrachtungen wird der übliche Wert einer Länge von 2 Byte angenommen. Er kann jedoch in der praktischen Anwendung auch länger sein.

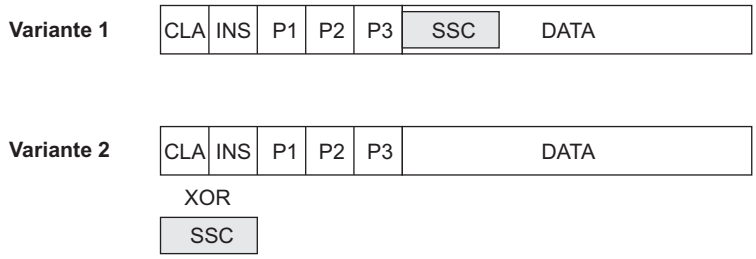

Bild 8.21 Die beiden Varianten eines Sendefolgezählers innerhalb einer Kommando-APDU. Variante 1 zeigt einen Sendefolgezähler SSC (*send sequence counter*) als TLV-kodiertes Datenobjekt im Datenteil. Bei Variante 2 wird ein Sendefolgezähler nur für die CCS-Berechnung durch XOR mit den APDU-Daten verknüpft und anschließend die ursprüngliche APDU übertragen.

Es gibt zwei grundsätzliche Varianten der Unterbringung des Sendefolgezählers in Kommando- und Antwort-APDUs. Er kann direkt als Zahlenwert in einem Datenobjekt in die jeweilige APDU eingefügt werden. Alternativ dazu kann man den Sendefolgezähler mit der gleichen Anzahl von Daten in der APDU mit XOR verknüpfen. Dann berechnet man die kryptografische Prüfsumme und restauriert anschließend die veränderten Daten wieder. Dem Empfänger dieser APDU ist der Sollwert des Sendefolgezählers bekannt, und er kann damit die APDU in gleicher Weise wie der Sender verändern. Anschließend berechnet er die kryptografische Prüfsumme und prüft die Richtigkeit der empfangenen APDU.

Während der Kommunikation findet dabei der folgende Ablauf statt: Zuerst fordert das Terminal einen Initialwert für den Zähler von der Chipkarte an. Diese gibt daraufhin eine zwei Byte lange Zufallszahl an das Terminal zurück. Das Terminal sendet nun das erste der sicherheitsrelevanten Kommandos mit dem Sendefolgezähler zur Chipkarte. Als Absicherung des Zählers und des Body kann das Authentic- oder das Combined-Verfahren verwendet werden. Die Chipkarte erhält nun die gesicherte APDU, prüft zuerst, ob Authentic- oder Combined-Verfahren einen Angriff indizieren, und vergleicht dann den Sendefolgezähler. Stimmt er mit dem in der Karte gespeicherten überein, so wurde während der Übertragung keine APDU eingeschoben oder ausgelassen.

Wie man deutlich erkennt, wird die Verwendung eines Sendefolgezählers nicht erst interessant, wenn mehrere Kommandos in einer bestimmten Reihenfolge ablaufen müssen, sondern auch schon bei einem einzelnen Kommando, da die Sitzung durch den Sendefolgezähler individualisiert wird. Schutz bietet der Zähler vor allem gegen Wiedereinspielung bereits gesendeter APDUs oder vor dem Ausschneiden von APDUs.

Wird der Sendefolgezähler zusammen mit dem Combined-Verfahren eingesetzt, so erhält jeder verschlüsselte Block ein anderes Aussehen, was als „diversity“ bezeichnet wird. Dies beruht darauf, dass der Sendefolgezähler beim Austausch jeder APDU erhöht wird und bei einem guten Verschlüsselungsalgorithmus die Veränderung eines Bits im Klartext Auswirkungen auf das Aussehen des ganzen Schlüsseltext-Blocks hat.

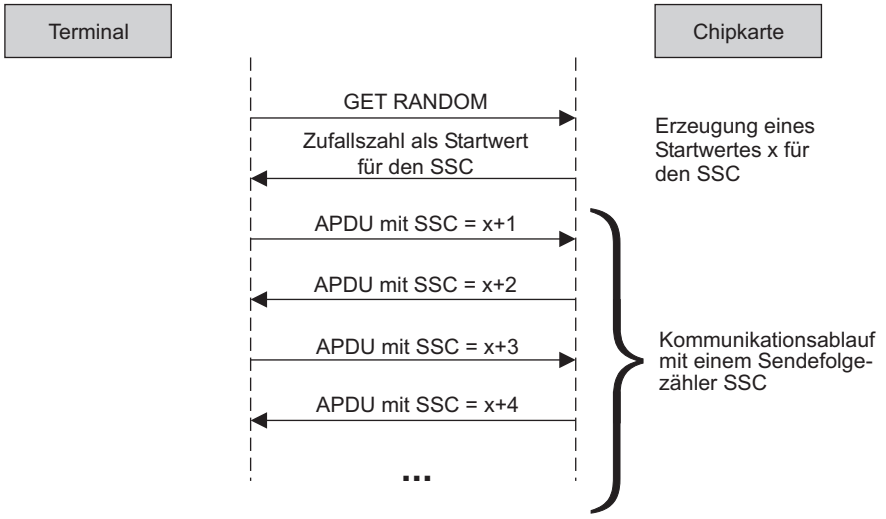

Bild 8.22 Übertragung von APDUs mit einem Sendefolgezähler SSC (*send sequence counter*).

8.5 Logische Kanäle (*logical channels*)

In Chipkarten mit mehreren unabhängigen Anwendungen ist vorgesehen, dass diese über so genannte logische Kanäle (*logical channels*) angesprochen werden. Diese ermöglichen es, dass bis zu 19 Anwendungen auf einer Chipkarte parallel mit einem Terminal Daten austau-

schen können. Benutzt wird nach wie vor die nur einmal vorhandene serielle Schnittstelle, doch auf logischer Ebene können die Anwendungen einzeln adressiert werden.[1]

Die Unterscheidung, welches Kommando nun zu welcher Anwendung gehört, wird aus Gründen der Rückwärtskompatibilität zu älteren Versionen der ISO/IEC 7816-4 zweistufig vorgenommen. Durch zwei Bits im Class-Byte (Bit 1 und Bit 2) lassen sich vier logische Kanäle adressieren, und es können so bis zu vier Sitzungen parallel mit Anwendungen in der Chipkarte ablaufen.[2] In der weiteren Stufe ist festgelegt, dass durch Benutzung eines besonderen Class-Bytes (Bit 8 = 0 und Bit 7 = 1) die niederwertigen vier Bits (Bit1 bis Bit 4) den logischen Kanal adressieren. Allerdings gibt es beim Kommunikationsablauf mit den verschiedenen Anwendungen auf der Chipkarte eine Einschränkung. Die von außen auf die Karte zugreifenden Prozesse müssen untereinander synchronisiert sein und dürfen keine Verschränkung aufweisen, da die Antwort-APDU der Karte keine Angabe des ursprünglichen logischen Kanals enthält. Es lässt sich also von außen nicht feststellen, welche Antwort welchen Returncode zurückgesendet hat. Das Resultat dieser fehlenden Kanalangabe ist, dass nach jedem Kommando an die Karte auf die daran anschließende Antwort gewartet werden muss.

Das Einsatzgebiet dieses mächtigen Mechanismus ist die gleichzeitige Benutzung von mehreren Anwendungen. Folgendes Beispielszenario soll dies verdeutlichen: Ein Benutzer führt ein Telefongespräch und möchte parallel dazu mit seinem Mobiltelefon dem Gesprächspartner ein Dokument in sicherer Weise zusenden. Der erste logische Kanal wird für die GSM-Anwendung geöffnet und wird dort unter anderem zur zyklischen Authentisierung der SIM gegenüber dem Mobilfunknetz benutzt. Der zweite logische Kanal wird zu einer anderen Anwendung auf der gleichen Multiapplikations-Chipkarte geöffnet, die alle Mechanismen und Schlüssel für eine gesicherte Kommunikation mit anderen Teilnehmern bereitstellt. Typischerweise würde dies eine WIM-Anwendung[3] sein. Ein anderes Beispiel in diesem Kontext wäre etwa der sichere Transfer von elektronischen Geldeinheiten parallel zu einem Mobilfunkgespräch.

So nützlich logische Kanäle erscheinen, so aufwändig ist ihre Verwaltung für das Chipkarten-Betriebssystem. Jeder logische Kanal bedeutet im Prinzip nichts anderes als eine völlig eigenständige Chipkarte mit allen Zuständen und Bedingungen. Dies bedeutet für das Betriebssystem, alle Informationen über mehrere parallel verlaufende Sitzungen gleichzeitig im Speicher zu verwalten. Der Aufwand dazu ist nicht zu unterschätzen und erfordert Mikrocontroller mit viel RAM. Fordert man zusätzlich Secure Messaging und alle Varianten der Authentisierung für jeden logischen Kanal individuell, steigt der Speicherbedarf sehr schnell in Bereiche, die nur von leistungsfähigen Mikrocontrollern für Chipkarten abgedeckt werden können.

8.6 Logische Protokolle

Neben den seit Jahren eingeführten Übertragungsprotokollen wie T=0 und T=1 findet man auch seit dem Beginn des neuen Jahrtausends die in der Internet-Welt etablierten Protokol-

[1] Siehe auch Abschnitt 11.12 „Kommandos für Datenübertragung“ auf Seite 431.
[2] Siehe auch Abschnitt 8.3.1 „Struktur der Kommando-APDUs“ auf Seite 239.
[3] Siehe auch 19.6 „Die WIM“ auf Seite 927.

le TCP/IP und HTTP bei Chipkarten. Die Verbreitung ist noch gering, doch werden sich diese weit verbreiteten Protokolle im Rahmen der immer größer werdenden Integration von Chipkarten in die üblichen IT-Infrastrukturen immer mehr etablieren. Die folgenden beiden Abschnitte enthalten eine überblickshafte Charakterisierung der relevanten Protokolle. Für Details sei hier auf die einschlägige Literatur verwiesen.

8.6.1 TCP/IP-Protokoll

Die beiden Protokolle TCP (*Transmission Control Protocol*) und IP (*Internet Protocol*) bilden zusammen eine der Grundlagen der Kommunikation im Internet. Die Entwicklung beider Protokolle fand in den siebziger Jahren des letzten Jahrhunderts statt. Den dazugehörigen Protokollstapel mit den vier Ebenen Anwendungs-, Transport-, Vermittlungs- und Netzzugangsschicht zeigt Bild 8.23.

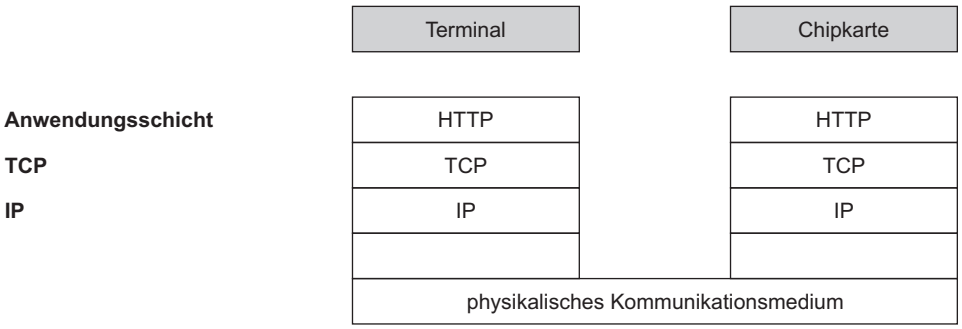

Bild 8.23 Der Protokollstapel TCP/IP mit darauf aufsetzendem HTTP-Protokoll. Die Schicht zwischen IP und physikalischem Kommunikationsmedium ist abhängig vom jeweiligen Betriebssystem.

Das IP-Protokoll gehört im Schichtenmodell der Internetprotokolle zur Internetschicht (*Internet layer*). Es ist ein verbindungsloses Protokoll und arbeitet unzuverlässig, was bedeutet, dass fehlerhafte Datenpakete beispielsweise nicht erneut angefordert werden. Aufgabe des IP-Protokolls ist die eindeutige Adressierung und Gruppierung von Rechnern in einem Netzwerk, die Weitervermittlung von Datenpaketen und die Wahl des Weges dieser Datenpakete (*routing*). Hauptsächlich wird heute Version 4 eingesetzt, welche in RFC 791 spezifiziert ist und oft als IPv4 abgekürzt wird. Der Adressraum hat dabei die Größe von 2^{32} und wird in vier dreistelligen Dezimalzahlen dargestellt. Da der 32 Bit-Adressraum von IPv4 mittlerweile in vielen Regionen zu klein geworden ist, findet langsam eine Umstellung auf das kompatible IP-Protokoll V6 statt. Es nutzt einen 48 Bit-Adressraum, ist in RFC 2460 spezifiziert und wird als IPv6 bezeichnet. Beide Versionen finden bei Chipkarten Einsatz.

Die unter der IP-Schicht befindliche Netzzugangsschicht kann, wie im Übrigen auch im Internet, vielgestaltig sein. Dies kann im einfachsten Falle die Kapselung der IP-Pakete in den Datenteil von APDUs und die anschließende Übertragung durch spezielle Kommandos mit T=0/1 sein. Üblicherweise wird jedoch die TCP/IP-Anbindung einer Chipkarte über USB und einen entsprechenden Gerätetreiber realisiert. Im Telekommunikationsbereich

wird außerdem das BIP-Protokoll[1] nach TS 102 223 verwendet, das eine direkte TCP/IP-Kommunikation mit der Chipkarte über die Luftschnittstelle zulässt.

TCP ist ein zuverlässiges Protokoll, das eine fehlerlose Kommunikation von einem Rechner zum anderen ermöglicht. Es arbeitet verbindungsorientiert und baut einen virtuellen Kanal zwischen zwei Endpunkten auf. Daten können in beide Richtungen übertragen werden. TCP ist damit der Transportschicht (*transport layer*) zuzuordnen. Spezifiziert ist TCP im RFC 793 und setzt üblicherweise auf dem IP-Protokoll auf. Die über TCP liegende Schicht erhält durch das Protokoll eine virtuelle Vollduplex-Verbindung, bei der unter anderem Übertragungsfehler automatisch durch erneute Anforderung der betroffenen Datenpakte korrigiert werden.

Die Festlegung einer Verbindung mit TCP findet über zwei Endpunkte statt, die sich jeweils wiederum aus der IP-Adresse und dem Port zusammensetzen. Dies wird auch als Socket bezeichnet und bildet die Schnittstelle zur darüber befindlichen Schicht wie beispielsweise HTTP. Ein Port wird durch eine 16 Bit-Zahl eindeutig gekennzeichnet. Eine TCP-Verbindung wird damit eindeutig über die beiden IP-Adressen von Quelle und Ziel und die beiden dazugehörigen Port-Adressen gekennzeichnet.

8.6.2 HTTP-Protokoll

Das HTTP-Protokoll (*hypertext transfer protocol*) ist im Internetprotokollstapel der Anwendungsschicht zuzuordnen und in der RFC 1945 (Hypertext Transfer Protocol – HTTP/1.0) und RFC 2616 (Hypertext Transfer Protocol – HTTP/1.1) spezifiziert. Es kann im Prinzip zum Transfer beliebiger Daten verwendet werden, wobei das dominierende Haupteinsatzgebiet die Übertragung von HTML-Seiten im Internet ist. Die Einordnung im Stapel der Internetprotokolle zeigt Bild 8.23 auf der vorherigen Seite.

Das HTTP-Protokoll muss für seine Funktion auf ein zuverlässiges darunterliegendes Protokoll aufbauen, ein Protokoll also, welches der darüberliegenden Schicht fehlerfreie Datenblöcke zur Verfügung stellt. Nahezu ausschließlich wird dafür das TCP-Protokoll verwendet. HTTP arbeitet als zustandsloses Protokoll, woraus unter anderem folgt, dass nach einer Datenübertragung die Verbindung zwischen den beiden Kommunikationspartnern nicht bestehen bleibt und Daten über die aktuelle Sitzung gegebenenfalls in der übergeordneten Schicht gespeichert werden müssen.

Die zwischen Server und Client ausgetauschten Datenblöcke bezeichnet man bei HTTP als Nachrichten (*message*). Der Client sendet dabei immer eine Anfrage (*request*) zu einem Server, der daraufhin eine Antwort (*response*) zurückgibt. Eine Nachricht setzt sich aus den Teilen Message Header und dem Message Body zusammen. Im Message Header befinden sich Verwaltungsinformationen für den Message Body, der die zu übertragenden Nutzdaten enthält.

Das Funktionsprinzip des HTTP-Protokolls ist dabei ziemlich einfach. Der Client sendet einen HTTP-Request an einen Server, der daraufhin die angeforderten Daten zurückgibt. Es handelt sich dabei um ein reines Master-Slave-Verhalten. Es besteht auch die Möglichkeit, mit einem entsprechenden Kommando Daten vom Client an den Server zu übertragen.

[1] Siehe auch Abschnitt 8.6.3 „BIP-Protokoll“ auf Seite 255.

Für die Anzeige von typischen HTML-Seiten benötigt man nur einige wenige HTTP-Kommandos. Mit dem Kommando GET fordert der Client Daten vom Server an, wie beispielsweise eine HTML-Seite. Ähnlich arbeitet POST, mit dem sich zusätzliche Daten an den Server übermitteln lassen. Mit PUT lassen sich Dateien vom Client auf den Server laden und mit DELETE löschen. Beide Kommandos werden im Internet selten genutzt, spielen aber bei einem Smart Card Web Server[1] durchaus eine Rolle.

Beim HTTP-Protokoll werden die Daten im Klartext übertragen, was bei vielen Anwendungen aus Sicherheitsgründen nicht tolerabel ist. Die Erweiterung HTTPS (*hypertext transfer protocol secure*) ermöglicht eine kryptografisch gesicherte Datenübertragung, die gegen Abhören und Manipulationen schützt. Dazu wird zwischen HTTP und dem darunter liegenden TCP eine zusätzliche Protokollschicht eingeführt. Diese heißt TLS (*transport layer security*) und ist in RFC 2246 spezifiziert. Der Vorgänger von TLS heißt SSL (*secure socket layer*), weshalb diese Schicht zur kryptografischen Absicherung des Datentransfers oft auch SSL/TLS genannt wird. Die RFC 2818 spezifiziert HTTP über TLS. Die üblicherweise bei SSL/TLS verwendeten Kryptoalgorithmen sind MD5, SHA-1, Triple-DES, AES und RSA.

8.6.3 BIP-Protokoll (*bearer independent protocol*)

Die einzige ursprüngliche Möglichkeit der Datenübertragung zwischen SIM-Karte und Hintergrundsystem waren Kurznachrichten (SMS). Ende der neunziger Jahr war klar, dass die daraus resultierende Übertragungsgeschwindigkeit bei weitem nicht für die künftigen Anforderungen ausreichen wird. Aus diesem Grund spezifizierte man das BIP-Protokoll (*bearer independent protocol*), das auch in der TS 102 223 normiert ist.

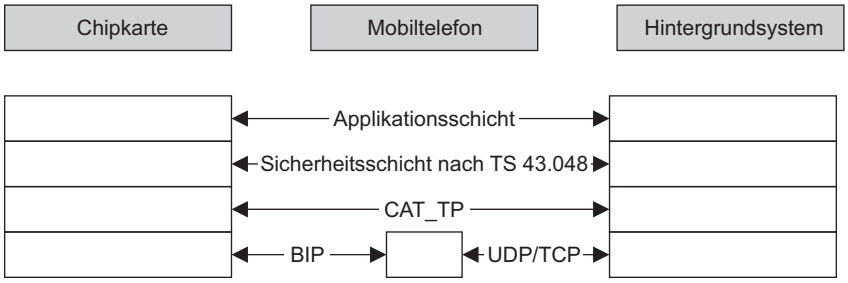

Bild 8.24 Überblick über die Zusammenhänge der Übertragungsprotokolle zwischen SIM, Mobiltelefon und Hintergrundsystem bei BIP und CAT_TP.

BIP basiert auf den proaktiven Kommandos[2] OPEN CHANNEL, CLOSE CHANNEL, SEND DATA, RECEIVE DATA sowie GET CHANEL STATUS und ermöglicht es einer im Mobiltelefon befindlichen Chipkarte über die Träger CSD, GPRS, IrDA oder Bluetooth mit einem verbundenen System auszutauschen. Das BIP-Protokoll beschränkt sich auf die Schnittstelle SIM und Mobiltelefon, die Kommunikation über die Luftschnittstelle mit einem Hintergrundsystem kann dann mit UDP (*user datagram protocol*) nach RFC 768 oder TCP vonstatten gehen, wie es auch Bild 8.24 zeigt.

[1] Siehe auch Abschnitt 23.3 „Smart Card Web Server“ auf Seite 992.
[2] Siehe auch Abschnitt 19.4.4.6 „SIM Application Toolkit“ auf Seite 904.

Aufbauend auf BIP wurde auch eine Übertragungsschicht spezifiziert, die eine direkte Kommunikation zwischen SIM-Karte und Hintergrund gestattet und auf BIP aufsetzt. Diese hat den Namen CAT_TP (*card application toolkit transport protocol*). CAT_TP lässt den Aufbau einer Verbindung sowohl seitens der Chipkarte als auch vom Hintergrundsystem zu und ist damit gut geeignet, SIM-Karten in ein größeres System einzubinden. Zur kryptografischen Sicherung der Verbindung kann man eine weitere Protokollebene benutzen, die in TS 43.048 spezifiziert ist. Damit lassen sich die zu übertragenden Daten sowohl authentisch als auch verschlüsselt in beide Richtungen übertragen. Das BIP-Protokoll ist außerdem die Grundlage für die Anbindung eines Smart Card Web Server[1] an den HTML-Browser des Mobiltelefons.

8.7 Anbindung von Terminals an übergeordnete Systeme

Um Chipkarten im Umfeld von PCs einsetzen zu können, bedarf es eines daran angeschlossenen Terminals und der Unterstützung durch die Software des PCs. In der Vergangenheit mussten für jedes Terminal eigene Softwaretreiber auf dem PC installiert werden. Diese hatten zudem ihre eigenen Softwareschnittstellen, so dass es praktisch nicht möglich war, eine terminalunabhängige Anwendungssoftware zu erstellen. Mitte der 90er begannen in verschiedenen Ländern und ausgehend von unterschiedlichsten Institutionen Spezifikationsarbeiten für Schnittstellen zur terminalunabhängigen Einbindung von Chipkarten in PC-Programme. Weltweit haben sich die beiden Industriestandards PC/SC (*personal computer/smart card*) und OCF (*open card framework*) etabliert. In Deutschland wie in einigen anderen Ländern gibt es seit langem die MKT-Spezifikation (Multifunktionales Kartenterminal – MKT), die im deutschsprachigen Raum eine überraschend hohe Verbreitung kennt. Alle drei Spezifikationen sind im Folgenden im Überblick dargestellt und können ansonsten auch via Internet kostenlos bezogen werden.

8.7.1 PC/SC

Ab Mai 1996 begann man, eine internationale Spezifikation für die Verbindung zwischen Karte und PC zu erstellen. Die Firmen Bull, Hewlett-Packard, Microsoft, Schlumberger, Siemens Nixdorf, Gemplus, IBM, Sun, Verifone und Toshiba arbeiteten an dieser Spezifikation mit.

Im Dezember 1997 wurde die Version 1.0 einer aus acht Teilen bestehenden „Interoperability Specification for ICCs and Personal Computer Systems“ veröffentlicht. Der Name für die Arbeitsgruppe war PC/SC (*personal computer/smart card*), welcher auch als Kurzform für die Spezifikation benutzt wird. Sie ist via Internet über den WWW-Server der Spezifikationsgruppe erhältlich [PC/SC].

Die PC/SC-Spezifikation ist zumindest ansatzweise plattformunabhängig, da sie auf allen Windows-unterstützten Rechnern funktioniert, welche den Großteil aller PCs ausmachen. Sie ermöglicht die Einbindung von Chipkarten in beliebige Anwendungen weitgehend unabhängig von einer Programmiersprache, da verbreitete Sprachen wie C, C++, C#, Java und

[1] Siehe auch Abschnitt 23.3 „Smart Card Web Server“ auf Seite 992.

Basic von PC/SC unterstützt werden. Die Voraussetzung ist lediglich, dass für das jeweils benutzte Terminal ein passender Treiber vorhanden und die verwendete Chipkarte PC/SC-kompatibel ist. Diese geforderte Kompatibilität ist jedoch relativ unkritisch, da der Rahmen verhältnismäßig weit gehalten wurde.

Tabelle 8.34 Die acht Teile der PC/SC-Spezifikation im Überblick.

PC/SC-Spezifikation	Inhalt
Teil 1: Introduction and Architecture Overview	Dieser Teil ist die Grundlage für alle folgenden Spezifikationsteile. Er enthält die zum Thema relevanten Normen, einen Überblick über die Systemarchitektur und die benutzten Hard- und Softwarekomponenten sowie Definitionen und Abkürzungen.
Teil 2: Interface Requirements for Compatible IC Cards and Readers	Festlegung der physikalischen Eigenschaften der kontaktbehafteten Chipkarte. Fixierung der grundlegenden elektrischen Eigenschaften wie Spannungsversorgung und Reset-Verhalten. Ebenso sind die Datenelemente, der Aufbau und die möglichen Abläufe von ATR und PPS definiert. Überblicksartig werden die Grundlagen der Datenübertragung auf physikalischer Ebene und die beiden Übertragungsprotokolle T=0 und T=1 beschrieben.
Teil 3: Requirements for PC-Connected Interface Devices	Aufstellung der Anforderungen an das Terminal sowie der unterstützten Terminaleigenschaften (Display, Tastatur u. Ä.).
Teil 4: IFD Design Considerations and Reference Design Information	Informationen zum Design von Terminals in Hinblick auf PS/2-Tastaturschnittstellen und USB-Schnittstellen.
Teil 5: ICC Resource Manager Definition	Detaillierte softwaretechnische Beschreibung des ICC Ressource Managers inklusive der dazugehörigen Klassen.
Teil 6: ICC Service Provider Interface Definition	Detaillierte softwaretechnische Beschreibung des ICC Service Providers und Crypto Service Providers inklusive der dazugehörigen Klassen.
Teil 7: Application Domain and Developer Design Considerations	Beschreibung der Benutzung der PC/SC-Spezifikation von Anwendungen aus.
Teil 8: Recommendations for ICC Security and Privacy Devices	Zusammenstellung und Festlegung von empfohlenen Funktionen und Mechanismen, welche eine PC/SC-Chipkarte unterstützen sollte. Dies schließt ein: das Dateisystem (MF, DF, EF), die dazugehörigen Dateizugriffsbedingungen, notwendige Systemdateien in der Chipkarte (Schlüssel, PIN u. ä.), Kommandos, Returncodes und kryptografische Algorithmen.

Ausgehend von den definierten Hard- und Softwarekomponenten lässt sich am unkompliziertesten die Übersicht über die PC/SC-Spezifikation gewinnen. Im folgenden sind die sieben Komponenten in ihren Funktionen und Schnittstellen zueinander beschrieben: ICC-Aware Application, ICC Service Provider, Crypto Service Provider, ICC Resource Manager, IFD Handler, IFD und ICC. PC/SC unterstützt in der aktuellen Version auch die Anbindung von Terminals für die kontaktlose Datenübertragung nach ISO/IEC 14443 Typ A und B.

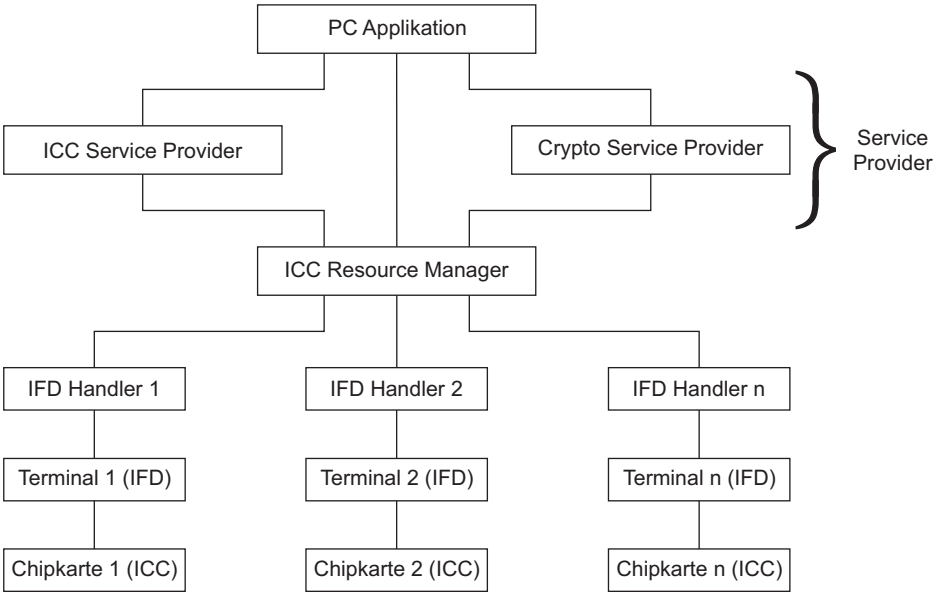

Bild 8.25 Überblick über die Softwarearchitektur der PC/SC-Spezifikation zur Anbindung von Chipkarten an PC-Betriebssysteme.

8.7.1.1 ICC-Aware Application

Diese Softwarekomponente ist die Anwendung, welche auf einem PC läuft und Funktionen und Daten einer oder mehrerer Chipkarten nutzen möchte. Sie berücksichtigt, dass diese Anwendung auch unter einem Mehrbenutzer-Betriebssystem mit Multitasking und Multithreading ausgeführt werden kann.

8.7.1.2 Service Provider

Der Service Provider dient dazu, die einzelnen Funktionen einer Chipkarte – unabhängig vom jeweiligen Betriebssystem der Chipkarte – abzukapseln. So kann beispielsweise über das API (*application programming interface*) des Service Providers eine Datei selektiert werden, ohne dass man das Chipkarten-Kommando oder überhaupt die Kodierung dazu kennen muss.

Die Komponente Service Provider ist in einen ICC Service Provider und einen Crypto Service Provider zweigeteilt. Der Grund dafür war, dass die in vielen Ländern bestehenden Exportbeschränkungen für kryptografische Algorithmen umgangen werden können, indem man den exportrechtlich problematischen Crypto Service Provider abtrennt und nicht mitliefert. Die PC/SC-Schnittstelle würde dann bis auf alle Kryptofunktionen nach wie vor benutzbar bleiben.

Der Service Provider muss nicht ein aus einem Stück bestehendes Programmteil sein, sondern kann aus über Netzwerke verbundenen Softwarekomponenten bestehen. Damit könnte zum Beispiel der Crypto Service Provider auf einem kryptografisch sicheren oder sehr

leistungsfähigen Rechner weit entfernt von den restlichen PC/SC-Bestandteilen ausgeführt werden.

8.7.1.3 ICC Resource Manager

Der ICC Resource Manager ist die wichtigste Komponente der PC/SC-Architektur. Er ist der Verwalter von allen Ressourcen, die für die Einbindung von Chipkarten in das Betriebssystem notwendig sind. Drei fundamentale Aufgaben hat der ICC Resource Manager zu erfüllen:

Er ist verantwortlich für die Erkennung der angeschlossenen Terminals sowie der Chipkarten. Zudem muss er auch noch das Stecken und Ziehen einer Chipkarte im Terminal bemerken und daraufhin eine diesbezügliche Meldung zur Verfügung stellen.

Als zweite Aufgabe muss er die Belegung von Terminals durch eine oder mehrere Anwendungen verwalten. Dazu kann die Ressource „Terminal“ ausschließlich an eine bestimmte Anwendung vergeben werden. Sollten jedoch mehrere Anwendungen gleichzeitig auf ein einzelnes Terminal zugreifen, muss dieses vom ICC Resource Manager als aufgeteilt (*shared*) gekennzeichnet und verwaltet werden.

Die dritte Funktion ist die Bereitstellung von Transaktionsprimitiven (*transaction primitives*). Dabei werden die für eine bestimmte Funktionalität relevanten Kommandos zu einer Gruppe verbunden. Damit ist sichergestellt, dass sie unmittelbar nacheinander in Folge ausgeführt werden. Sonst könnte es passieren, dass zwei Anwendungen zeitgleich und zueinander unkoordiniert mit ihren jeweiligen Kommandos auf eine Chipkarte zugreifen.

Die dabei zum Tragen kommende Problematik kann man am einfachsten anhand des folgenden Beispiels veranschaulichen: In einer Chipkarte kann zu einer Zeit nur jeweils eine Datei selektiert sein. Versuchen zwei Anwendungen nun gleichzeitig mit Selektion (SELECT) unterschiedlicher Dateien und anschließendem Lesekommando (z. B.: READ BINARY), Daten aus der Chipkarte abzurufen, dann ist undefiniert, welche Datei nun wirklich ausgelesen wird. Dies hängt einzig und allein von der Reihenfolge des Eintreffens der Kommandos bei der Chipkarte ab. Weitaus komplizierter, doch nicht minder verzwickt wird die Situation, wenn komplexe Abläufe, wie etwa beim Bezahlen mit einer elektronischen Geldbörse, zwischen mehreren Anwendungen und einer Chipkarte durchgeführt werden müssen. Der ICC Resource Manager sorgt dafür, dass es nicht zu Überschneidungen zusammengehöriger Kommandosequenzen kommen kann, sondern die einzelnen Abläufe nacheinander ausgeführt werden.

8.7.1.4 IFD Handler

Der IFD Handler ist eine Art von Treiber, der für das jeweilige Terminal spezifisch ist. Er hat die Aufgabe, das Terminal an die festgelegten Schnittstellen des PCs anzubinden und die individuellen Eigenschaften des Terminals auf die PC/SC-Schnittstelle abzubilden. Der IFD Handler stellt damit sozusagen einen datentechnischen Kanal vom PC zu einem speziellen Terminal her.

8.7.1.5 IFD (*interface device*)

Die Komponente IFD der PC/SC-Spezifikation ist ein Terminal, das über eine Schnittstelle an den PC angeschlossen ist. Diese kann beliebig sein, das Terminal kann also beispielsweise über RS-232, USB oder PC-Card (ex PCMCIA) mit dem Computer verbunden sein. Das Terminal muss die ISO/IEC 7816-1, 2, 3-Normen erfüllen, was unter anderem bedeutet, dass beide asynchrone Übertragungsprotokolle T=0 und T=1 unterstützt werden müssen. Optional können zusätzlich die synchronen Übertragungsprotokolle (2-Draht, 3-Draht, I^2C-Bus) für Speicherkarten nach ISO/IEC 7816-10 implementiert sein.

Neben einem Display werden beim Terminal für die Benutzeridentifizierung auch numerische Tastatur, Fingerabdruckscanner und andere biometrische Identifikationssensoren durch PC/SC unterstützt.

8.7.1.6 ICC (*integrated chip card*)

Die von der PC/SC-Spezifikation obligatorisch unterstützten Chipkarten sind Prozessorkarten und kompatibel zu den Normen ISO/IEC 7816-1, 2, 3. Falls es das Terminal zulässt, können Speicherkarten nach ISO/IEC 7816-10 eingebunden werden.

8.7.2 OCF

Im Jahr 1997 wurde die Open Card Initiative [OCF] von mehr als zehn Firmen aus dem Umfeld von Chipkarten und PCs gegründet. Der Zweck war, eine Schnittstelle zu Chipkarten auf PCs zu schaffen, die unabhängig vom Betriebssystem des PCs (Windows, Unix, ...) und unabhängig von der jeweiligen Anwendung auf der Chipkarte ist. Das Ergebnis ist OCF (*open card framework*), eine Java-basierte Schnittstelle auf PCs, mit deren Hilfe von PC-Programmen aus auf Chipkarten-Anwendungen zugegriffen werden kann. OCF hat sich mittlerweile zu einem Industriestandard im Java-Umfeld etabliert.

OCF bietet auf Anwendungsebene dem Entwickler zwei grundsätzliche Möglichkeiten. Der ursprünglich präferierte Ansatz ermöglicht es, komplexe Kommandoseqeunzen an die Chipkarte zusammenzufassen. Diese können bereits vom Betriebssystemhersteller der Chipkarte oder auch vom Anwendungsentwickler bereitgestellt werden. Der Vorteil dabei ist, dass komplizierte Kommandos bzw. Kommandosequenzen wie sie beispielsweise bei elektronischen Geldbörsen auftreten, auf Anwendungsebene des PCs relativ einfach angestoßen werden können.

In der Praxis wird jedoch von dieser Variante eher verhalten Gebrauch gemacht, da man bei der Softwareentwicklung auf dem PC gerne die volle Kontrolle über die Kommandos an die Chipkarte hat. Deshalb nutzt man heute OCF vor allem, um von der Programmiersprache Java aus Kommandos direkt als APDU kodiert an die Chipkarte zu senden.

8.7.3 MKT

In Deutschland wurde bereits frühzeitig damit begonnen, eine Spezifikation zur softwaretechnischen Anbindung von Terminals an PCs zu erstellen. Dies ist die seit 1994 von Tele-

trust Deutschland [TTT] in mehreren Versionen veröffentlichte MKT-Spezifikation (Multifunktionale Karten Terminal – MKT). Sie ist vor allem auf die Belange des Gesundheitsbereiches ausgerichtet, wird jedoch mittlerweile auch von vielen anderen Terminals innerhalb Deutschlands als Grundlage benutzt.

Die MKT-Spezifikation setzt sich aus 7 Teilen zusammen. Teil 1 beinhaltet das MKT-Basiskonzept, das einen grundlegenden Überblick über die Softwarearchitektur und das MKT-Terminal enthält. In Teil 2, dem so genannten CT-ICC-Interface (*card terminal – integrated chip card*), ist die Schnittstelle für kontaktorientierte Chipkarten mit synchroner und asynchroner Übertragung spezifiziert.

Teil 3 enthält die Beschreibung einer anwendungsunabhängigen Schnittstelle für Terminals namens CT-API (Anwendungsunabhängiges Card Terminal Application Programming Interface). Das CT-API ist unabhängig von einer Programmiersprache und prozedural aufgebaut. Dabei stehen die folgenden 3 Funktionen zur Verfügung: „CT_init“ für die Initialisierung einer Verbindung, „CT_data“ für den Datenaustausch auf einer bestehenden Verbindung und „CT_close“ zum Schließen einer Verbindung.

Ergänzend dazu findet sich in Teil 4 die Festlegung einiger grundlegender und anwendungsunabhängiger Kommandos zur Steuerung von Terminals, welche die Bezeichnung CT-BCS (Anwendungsunabhängiges Card Terminal Basic Command Set) hat.

In Teil 5 sind für Chipkarten mit synchroner Datenübertragung der ATR und allgemeine Datenbereiche beschrieben. Teil 6 enthält die dazugehörigen Übertragungsprotokolle sowie entsprechende generelle Kommandos an das Terminal. Darauf aufbauend spezifiziert Teil 7 die Übersetzung von ISO/IEC 7816-4-Kommandos in Kommandos für Chipkarten mit synchroner Datenübertragung, d. h. Speicherkarten.

Die MKT-Spezifikation war weltweit eines der ersten Dokumente ihrer Art und wurde in Deutschland durch Tausende Terminals für 80 Millionen Krankenversichertenkarten auf eine enorm breite Basis gestellt. Obwohl technisch mittlerweile sicherlich nicht mehr auf dem neuesten Stand, wird diese Spezifikation noch für viele Jahre einen nationalen Industriestandard darstellen.

8.7.4 MUSCLE

Die Verwendung von Chipkarten unter Linux erfordert, wie bei allen anderen PC-Betriebssystemen, entsprechende Treiber. Diese waren anfangs nicht verfügbar, so dass es unter anderem aufwändig war, Chipkarten für einen Login unter Linux zu benutzen.

Im Jahr 2000 wurde die erste Version von MUSCLE (*Movement for the Use of Smart Cards in a Linux Environment*) veröffentlicht, die genau diese Lücke schloss. MUSCLE basiert seitens der Architektur stark auf PC/SC, liegt jedoch im Gegensatz zu PC/SC im Quellkode veröffentlicht vor [MUSCLE] und steht unter GPL-Lizenz so dass es auch von Dritten adaptiert und weiterentwickelt werden kann. Mit MUSCLE wurde eine API unter Linux geschaffen, die einen unkomplizierten Zugriff auf Chipkarten in einem angeschlossenen Terminal ermöglicht.