

die der eigentliche Interpreter ist. Aus einem anderen Blickwinkel betrachtet, ist die JVM eine Simulation des Java-Prozessors auf einem beliebigen Zielsystem. Der Zielprozessor arbeitet seinerseits dann selbstverständlich mit Native-Code. Der große Vorteil dieser Aufteilung besteht darin, dass nur die im jeweiligen Native-Code programmierte JVM auf den Zielprozessor portiert werden muss. Anschließend ist der Java-Bytecode auf dem neuen Prozessor lauffähig.

Da die Aufgabe der Virtual Machine zur Laufzeit nicht nur die stupide Interpretation von Bytecodes ist, sondern diese unter anderem auch Typprüfungen vornimmt und Objektzugriffe überwacht, heißt sie auch Sandbox. Dieses Synonym zeigt anschaulich, dass ein Java-Programm nur in seinem eigenen Sandkasten agieren kann und diesen auch nicht verlassen darf, da es sonst von der Virtual Machine in seinem Tun gestoppt wird.

Die kompilierten, d. h. in Bytecode übersetzten und mit zusätzlichen Informationen versehenen Java-Programme werden in einer so genannten Class-Datei (*class-file*) abgespeichert. Sie werden nach dem Laden von der Java Virtual Machine ausgeführt. Ein oder mehrere Class-Dateien ergeben dann ein Applet. Es beinhaltet eine komplette Chipkarten-Anwendung und besitzt auch einen eigenen Application Identifier (AID). Im Umfeld von Java für Chipkarten wird ein Applet auch manchmal Cardlet genannt.

Die Hardware-Unabhängigkeit von Java hat natürlich auch ihren Preis, der vor allem in einer im Vergleich zu den üblichen Programmiersprachen langsameren Ausführungsgeschwindigkeit besteht. Dieses Problem ist immer noch nicht zufrieden stellend gelöst, doch konzentrieren sich Weiterentwicklungen und Verbesserungen von Java vor allem auf dieses Gebiet. Ein Schritt, der bereits beschränkt wurde, ist der Einsatz so genannter Just-In-Time-Compiler (JIT-Compiler), die beim ersten Aufruf eines Java-Bytecodes diesen in die Maschinensprache des Prozessors übersetzen. Dies führt natürlich zu einer deutlichen Verlangsamung beim ersten Aufruf, doch werden alle folgenden Aufrufe dann wesentlich schneller abgearbeitet. Einen softwaretechnisch ziemlich aufwendigen JIT-Compiler auf einer Chipkarte zu realisieren, wird jedoch in den nächsten Jahren aufgrund von Speicherplatzmangel nicht möglich sein.

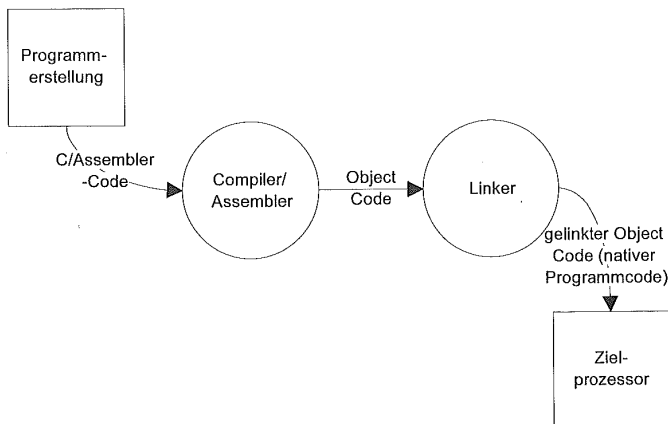


Bild 5.48 Datenflussdiagramm als Überblick des üblichen Ablaufs vom Source Code in C oder Assembler bis zum ausführbaren Maschinencode für den Zielprozessor.

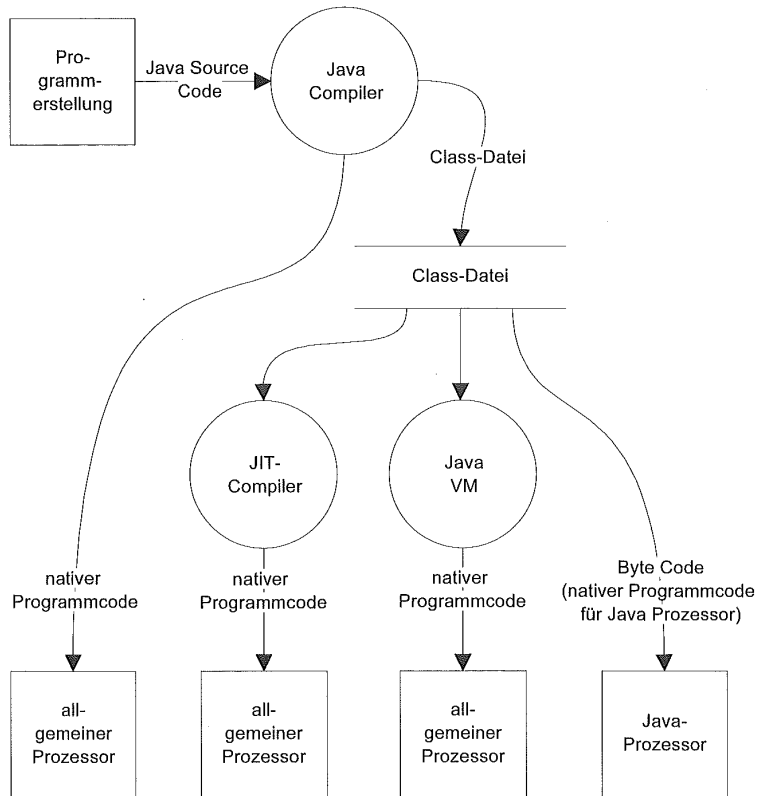


Bild 5.49 Datenflussdiagramm als Überblick der möglichen Abläufe vom Java Source Code bis zum ausführbaren Maschinencode für den Zielprozessor. Der links dargestellte Weg wird zwar von manchen Compiler-Herstellern angeboten, entspricht jedoch nicht der ursprünglichen Philosophie von Java, da die Hardware-unabhängigkeit nicht mehr gegeben ist.

Die direkte Kompilation von Java-Programmen in die Maschinensprache des Zielsystems würde in der heterogenen Welt der Chipkartenprozessoren, im Gegensatz zur PC-Welt, keinen Sinn ergeben. So kommt als letzter Weg nur noch der Einsatz spezieller Hardware in Frage.

Man könnte dabei so vorgehen, dass neben den bisherigen 8051- und 6805-Prozessor-kernen ein spezieller Javaprozessor auf dem Halbleiter integriert würde. Damit erhält man den Vorteil, die zeitkritischen Routinen (Datenübertragung, Kryptoalgorithmen) nach wie vor in Assembler programmieren zu können, doch zugleich für alle höheren Software-schichten eine Hochsprache wie Java zu benutzen.

Technisch wäre dies ohne weiteres möglich, wenn man sich Folgendes veranschaulicht: Der Java Chip von Sun (microJava 701) hat bei einem 0,25 µm Halbleiterprozess eine Seitenlänge von 7 mm (= 49 mm²), arbeitet mit 2,5 Volt Spannung bei 200 MHz, besitzt 2,8 Millionen Transistoren und hat eine Leistungsaufnahme von lediglich 4 Watt. Da dieser Chip jedoch viele Funktionen besitzt, die bei Chipkarten nicht notwendig sind (z. B.:

Gleitzalarithmetik, Ansteuerung externer Speicher), liegt es technisch durchaus im Bereich des Möglichen, einen auf Chipkarten-Anforderungen geschrumpften Java-Prozessor auf einem Chipkarten-Mikrocontroller unterzubringen.

Eine weitere Variante zur Lösung des Geschwindigkeitsproblems ist die Hardware-technische Erweiterung des Befehlssatzes der Chipkarten-Prozessoren. Mit diesem Lösungsansatz können etwa 80 Prozent der Maschinenbefehle der Java-VM durch den Prozessor abgedeckt werden, was technisch nicht allzu kompliziert ist und eine eklatante Geschwindigkeitserhöhung zur Folge hat. Dies ist zurzeit der bevorzugte Lösungsansatz vieler Hersteller von Chipkarten-Mikrocontrollern.

Tabelle 5.22 Die Ausführungsgeschwindigkeit von Programmen in Abhängigkeit von der Programmiersprache. Die Werte in der Tabelle sind Schätzungen für PCs und sollen lediglich zur Orientierung dienen. Referenz für den PC ist ein 300 MHz Pentium II und für den Javaprozessor der 200 MHz picoJava 701.

Programmiersprache	Ausführungsgeschwindigkeit
C/C++	1
Java, Programmausführung durch Interpreter	10 . . . 20
Java, Programmausführung durch Interpreter und Einsatz eines JIT-Compilers	≈ 5
Java, Programmcode kompiliert als Native-Code	1 . . . 2
Java, Programmausführung durch Javaprozessor	≈ 1,2

Ein anderer Weg der schnelleren Ausführung von Java-Bytecode eröffnet sich mit den 32-Bit-Chipkarten-Mikrocontrollern. Allerdings ist selbst mit 32-Bit-Prozessoren die Ausführungsgeschwindigkeit von interpretiertem Java-Programmcode immer noch um den Faktor 10 bis 20 langsamer als von Programmcode, geschrieben in C. Der große Vorteil liegt jedoch darin, dass diese Prozessoren allgemein einsetzbar sind.

Der Nachteil der mangelnden Abarbeitungsgeschwindigkeit von interpretiertem Programmcode gegenüber kompiliertem Programmcode lässt sich aber durch geeignete Programmierschnittstellen bei den Interpretern stark reduzieren. Diese APIs (*application programming interface*) schaffen die Möglichkeit, vom interpretierten Programmcode aus Programmteile in der Maschinensprache des jeweiligen Prozessors aufzurufen. Solche über ein API aufgerufene Native-Programme werden dann mit voller Prozessorgeschwindigkeit abgearbeitet.

So ideal dieser Weg der Geschwindigkeitserhöhung im ersten Ansatz auch klingen mag, so hat er durchaus auch seine Nachteile. Das API für Native-Code muss wohl überlegt sein, damit es auch allgemein eingesetzt werden kann und nicht nur für wenige Sonderfälle sinnvoll ist. Dies ist jedoch mit einiger Überlegung realisierbar. Der zweite Nachteil wiegt jedoch schwerer: Kompatibilität und Hardware-Unabhängigkeit einer Programmiersprache wie Java ist nur dann möglich, wenn alle APIs gleich sind. Existieren unterschiedliche APIs, entweder in Hinsicht auf Schnittstellen oder auf Funktionalität, dann wäre eine standardisierte Programmiersprache kein großer Vorteil. Man müsste dann für jede Plattform mit einem eigenen API diverse Anpassungen im Source-Code durchführen. Dies ist auch der Grund, warum im Java Card Forum sehr viel Arbeit in die Standardisierung der APIs gesteckt wird, die elementar für eine Plattformunabhängigkeit sind.

Die Java Virtual Machine (JVM)

Die Java Virtual Machine ist das wesentliche Element der Java-Technologie. Sie ist die Simulation eines Java-Prozessors und kann auf jedem genügend leistungsfähigen Prozessor in Software implementiert werden. Soll auf einem neuen Prozessor Java-Bytecode ausgeführt werden, so muss nur die Java Virtual Machine portiert werden. Diese ist üblicherweise in der Programmiersprache C geschrieben, sodass die eigentliche Portierung unter Umständen nur einige kleinere Anpassungen und eine Neukompilation des Source-Codes der Virtual Machine erfordern. Die Größe einer Java Virtual Machine auf einem PC bewegt sich im Bereich zwischen 100 kByte und 200 kByte.

Die Virtual Machine für Java besitzt all jene Komponenten, die auch Bestandteil eines realen Prozessors sind. So hat sie einen eigenen Befehlssatz, den Bytecode, und auch Register wie Programmzähler und Akku. Die zu verarbeitenden Daten werden der Virtual Machine als so genannte Class-Datei übergeben. Diese Datei enthält die festgelegten Konstanten, den auszuführenden Bytecode als Methoden und diverse zusätzliche Informationen.

Tabelle 5.23 Der Aufbau einer Class-Datei im Überblick

Datenelemente einer Class-Datei
Erkennung (<i>magic number</i>)
Versionsnummer
Konstantenpool
Methoden
Attribute von Klassen, Feldern und Methoden

Tabelle 5.24 Tabelle der Einschränkungen des Funktionalitätsumfangs von Java für Chipkarten gegenüber dem vollständigem Java.

Funktionalität	Java Card	Java
Operatoren	alle	alle
Funktionen der Ablaufkontrolle	ja	ja
Ausnahmebehandlung (<i>exceptions</i>)	ja	ja
Datentypen: boolean, byte, short	ja	ja
Datentyp: int	optional	ja
Datentypen: long, float, double, char	nein	ja
Felder für die unterstützten Datentypen	nur eindimensionale	auch mehrdimensionale
dynamisches Nachladen von Klassen	nein	ja
Cloning von Klassen	nein	ja
Felder von Objekten	nein	ja
Klonen von Objekten	nein	ja
dynamische Objekterzeugung	ja	ja
virtuelle Methoden	ja	ja
Pakete (<i>packages</i>)	ja	ja
Interfaces	ja	ja
dynamische Speicherverwaltung (Garbage Collection)	nein	ja
Threads	nein	ja

Der Bytecode von Java ist sehr platzsparend und in etwa so kompakt wie Maschinencode. Die Speicherplatz-Bilanz gegenüber nativem Maschinencode wird nur durch die bei Java zwangsläufig notwendige Virtual Machine verschlechtert. Der Unterschied wird natürlich umso größer, je kleiner der Programmcode im Verhältnis zur Virtual Machine ist.

Grundsätzlich ist Bytecode sehr ähnlich den Maschinenbefehlen für einen üblichen Prozessor. Es gibt beispielsweise Kommandos zur Stack-Manipulation, logische und arithmetische Operationen, Kommandos, um auf Register des virtuellen Java-Prozessors zuzugreifen, und sogar Zugriffsmöglichkeiten auf Felder. Eine ausführliche Beschreibung des Bytecodes und der Java Virtual Machine findet sich bei Tim Lindholm und Frank Yellin [Lindholm 97].

Tabelle 5.25 Tabelle der bei Java für Chipkarten vorhandenen Datentypen, ihres Platzbedarfs im Speicher und ihres Wertebereiches. Der Datentyp „int“ ist optional.

Datentyp	Größe	Wertebereich
boolean	1 Byte	true, false
byte	1 Byte	– 128 ... 127
short	2 Byte	– 32768 ... 32767
int	4 Byte	– 2 147 483 648 ... 2 147 483 647

Aufgrund der erheblich geringeren Systemressourcen eines Chipkarten-Mikrocontrollers mussten bei der Java Card VM einige Einschränkungen gegenüber der ursprünglichen VM für PCs getroffen werden. Sie besitzt keine Garbage Collection, welche nicht mehr benötigten Speicher automatisch wieder dem Freispeicher zuordnet. Die bei der Erstellung dieses Buches noch nicht verabschiedete Version 2.2 der Java Card Spezifikation sieht jedoch bereits einen optionalen Garbage Collector vor. Damit werden dann erzeugte Objekte bei Bedarf auch wieder gelöscht werden können. Die Unterstützung für die Class-Dateien musste ebenfalls stark eingeschränkt werden. Es stehen auf Chipkarten auch weniger Datentypen zur Verfügung, und auch der Bytecode selber wurde von 149 Befehlen auf 76 Befehle gekürzt.

Die oncard Java VM für Chipkarten hat einen Programmcodeumfang in der Größenordnung von etwa 40 kByte 8051 Maschinencode, wenn sie in der Programmiersprache C erstellt wird, wobei auf den reinen Bytecode-Interpreter etwa 12 kByte entfallen. Dazu werden noch etwa 400 Byte RAM benötigt. Das API mit den Klassen aus `java-card.framework` und `javacardx.framework` benötigt 3 bis 4 kByte Speicherplatz, wobei es aber zum Großteil in Java programmiert ist. Zusätzlich wird mindestens noch ein rudimentäres Betriebssystem mit Übertragungsprotokollen, Kryptoalgorithmen und vielen Hardware-nahen Funktionen benötigt. Der Codeumfang dafür bewegt sich bei Programmierung in Assembler zwischen 6 und 8 kByte [Baentsch 99]. Im Zusammenhang mit diesen Speicherplatzwerten muss jedoch angemerkt werden, dass diese sehr von den unterstützten Funktionen und vielschichtigen Randbedingungen abhängen. Deshalb können die oben genannten Werte im Einzelfall von Realisierung zu Realisierung erheblich abweichen.

Die Korrektheit dieser verhältnismäßig kleinen Programme ist von außerordentlicher Wichtigkeit, da ein Fehler oder eine Sicherheitslücke in der Java VM das gesamte Sicherheitskonzept von Java auf Chipkarten untergraben könnte. Das Design und die Implementierung müssen also weitgehend fehlerfrei sein. Zur Überprüfung dieser Eigenschaft werden üblicherweise Evaluierungen im Rahmen von ITSEC bzw. Common Criteria herange-

zogen. Der geringe Umfang an Programmcode für die Java VM erleichtert diese Prüfung natürlich erheblich, zumal es dadurch auch möglich ist, die vollständige Funktionalität der VM formal zu beschreiben. Typisch sind dabei Evaluierungen nach ITSEC E4.

Aufgrund der kleinen Speichergrößen bei Chipkarten war es notwendig, die Java VM in einen oncard-Teil und einen offcard-Teil aufzutrennen. Es ist ohne weiteres und ohne Einbußen an Leistungsfähigkeit oder Sicherheit möglich, die statischen Prüfungen außerhalb der Chipkarte in der Offcard-VM vorzunehmen. Die Verbindung zwischen den beiden Teilen der VM sind Daten im CAP-Format. Zur vollständigen Absicherung müssen diese kryptografisch geschützt werden, idealerweise mit einer digitalen Signatur, damit sie während der Übertragung nicht manipuliert werden können. Diese wäre nämlich für einen Angreifer ein Erfolg versprechender Ansatzpunkt, da er so unter Umständen die Sicherheitsmechanismen der Offcard-VM umgehen könnte.

Tabelle 5.26 Die Beschränkungen von Java für Chipkarten im Überblick. Diese weitgefassten Grenzen stellen bei der Softwareentwicklung für Chipkarten zurzeit keine Einschränkung dar.

Klassen	Eine instanzierte Klasse kann maximal 255 Felder besitzen. Eine Klasse kann maximal 256 statische Methoden besitzen. Eine Klasse kann maximal 256 statische Felder besitzen. Eine Klasse kann maximal 15 Interfaces besitzen.
Paket (<i>package</i>)	Ein Paket kann maximal 255 öffentliche Klassen und Interfaces besitzen.
Methoden	Eine Methode kann maximal 255 lokale Variablen besitzen. Eine Methode kann maximal 32 767 Bytecodes besitzen.
Arrays	Arrays können maximal 32 767 Felder besitzen.
Switch Befehl	Eine Switch-Befehl kann maximal 65 536 Verzweigungen unterstützen. Falls der Datentyp „int“ unterstützt wird, ist die maximale Anzahl von Verzweigungen im Switch-Befehl analog Java auf PCs und damit abhängig vom Wertebereich des gewählten Datentyps (char, byte, short, int).

Das Stack-orientierte Java benötigt erwartungsgemäß einen Stack und auch einen Heap. Dieser wird für jedes Applet auf der Chipkarte separat angelegt und verwaltet. Der Stack wird vor allem für die Datenübergabe bei den Methodenaufrufen benutzt, und der Heap dient als Ablage für die Objekte. Übliche Größen sind etwa 700 Byte für den Stack im RAM und etwa 5 bis 8 kByte für den Heap im EEPROM. Kleinere Applets können aber durchaus mit 50 bis 60 Byte Stack und einigen wenigen hundert Byte Heap auskommen.

Die Java VM besteht aus den vier Funktionseinheiten Bytecode Verifier, Loader, Interpreter und Security Manager.

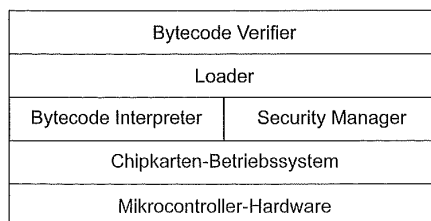


Bild 5.50 Die einzelnen Komponenten der Java Card Virtual Machine im Überblick.

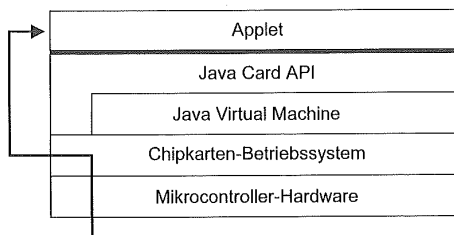
Tabelle 5.27 Die grundlegenden Funktionen in der Hauptschleife des Java Bytecode Interpreters

DO (	Hauptschleife des Interpreters
hole und merke Programmzähler	Holen und Merken des Programmzählers für den späteren Vergleich.
hole Opcode	
hole Operanden	
führe Maschinenbefehl aus	Führe den aus Opcode und Operanden bestehenden Maschinenbefehl für den virtuellen Java-Prozessor aus.
IF (Maschinenbefehl hat Programmzähler nicht geändert)	Falls der direkt vorher ausgeführte Java-Maschinenbefehl den Programmzähler nicht verändert hat (d. h. es war z. B. kein Bytecode für ein GOTO), setze den Programmzähler auf den nächsten Opcode.
THEN (erhöhe Programmzähler)	
) WHILE (Opcodes vorhanden)	Führe die Schleife so lange aus, bis alle Opcodes abgearbeitet sind.

Der Bytecode Verifier hat die Aufgabe, mannigfaltige statische Prüfungen an der übergebenen Class-Datei vorzunehmen. Das ist als Erstes eine Kontrolle des Dateiformats. Anschließend folgen Prüfungen des Konstantenpools, des Bytecodes auf syntaktische Korrektheit, Argumente der Methoden und Vererbungshierarchien der Objekte. Es existieren noch einige weitere Untersuchungen, die jedoch im Detail bei Frank Yellin [Yellin 96] beschrieben sind.

Nach dem Bytecode Verifier übernimmt der Loader die geprüften Daten und sendet sie im CAP-Format zum eigentlichen Interpreter in der Chipkarte. Die Daten sollten aus Sicherheitsgründen mit einer digitalen Signatur versehen sein, damit sie zwischen Loader und On-card-Teil der Java VM nicht manipulierbar sind. Der eigentliche Ladeprozess ist unabhängig von den Java Card-Spezifikationen. Hier hat sich die OP-Spezifikation¹⁶ als Industriestandard etabliert.

Nach dem Ladevorgang befindet sich der ausführbare Bytecode mit diversen weiteren Informationen im Speicher der Chipkarte und kann vom On-card-Teil der Java VM ausgeführt werden. Der eigentliche Interpreter liest Bytecode für Bytecode mit den jeweils dazugehörigen Argumenten und setzt diesen in native Maschinenbefehle des Zielprozessors um. Parallel zur Bytecode-Interpretation arbeitet der Security Manager, der unter anderem



Laden eines Applets

Bild 5.51 Der grundsätzliche Ablauf beim Laden eines Applets in ein Chipkarten-Betriebssystem mit Java.

¹⁶ Siehe auch Abschnitt 5.11 „Open Platform“.

ständig die Einhaltung von Feld-, Stack- und Heapgrenzen prüft. Er ist bei einer Verletzung der festgelegten Sicherheitsgebote ermächtigt, augenblicklich eine Ausnahmebehandlung (*exception*) zu starten und die Abarbeitung des problematischen Bytecodes zu stoppen.

Java für Chipkarten

Die wirklichen großen Vorteile einer modernen Programmiersprache wie Java im Einsatz bei Chipkarten sind nicht nur in der Richtung zu sehen, dass nun jeder Programme für Chipkarten schreiben kann. Das wäre mit Assembler oder C, offen gelegten Schnittstellen und einigen Anpassungen mit den meisten Chipkarten-Betriebssystemen denkbar. Interessant ist ein Konzept wie die Java Card auch für große Systembetreiber. Diese haben das Problem, dass sie unterschiedliche Chipkarten-Betriebssysteme auf unterschiedlichen Mikrocontrollern von unterschiedlichen Kartenherstellern einkaufen müssen. Dieses Multiple-Sourcing, das aus taktischen Gründen (reduzierte Abhängigkeit von einem Hersteller, Preisdruck auf die Hersteller) durchaus Sinn ergibt, verursacht aber andererseits laufend Probleme mit Kompatibilität und Testing. Es ist auf absehbare Zeit nicht erreichbar, dass sich zwei Betriebssysteme unterschiedlicher Hersteller in allen Variationsmöglichkeiten auf der Schnittstelle gleich verhalten. Dies stellt jedoch für Systembetreiber eine ernsthafte Schwierigkeit dar.

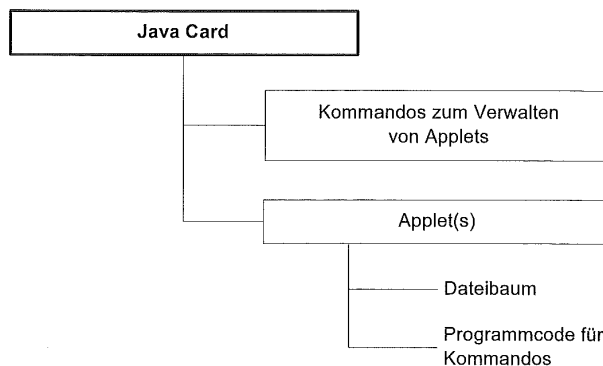


Bild 5.52 Die beiden grundlegenden informationstechnischen Bestandteile einer Chipkarte mit Java.

Die ideale Lösung aus Sicht der Systembetreiber wäre somit Hardware-unabhängiger Programmcode, der auf einem evaluierten Interpreter in Chipkarten einheitlich ausgeführt wird. Man müsste dann dieses Anwendungsprogramm nur noch einmal erstellen, testen, evaluieren, und es könnte dann von den verschiedensten Chipkarten-Betriebssystemen abgearbeitet werden. Nach außen, d. h. auf der Schnittstelle, wären keine Unterschiede sichtbar. Damit blieben die Vorteile des Multiple-Sourcing erhalten, während die Nachteile verschwinden.

Diese Hinweise sollte man im Hinterkopf behalten, wenn man betrachtet, wie Java in Chipkarten eingebunden ist. Diese ersten Versionen sahen noch vor, dass der Java Bytecode in einem EF unter dem MF oder einem DF abgelegt wird. Mit einem EXECUTE Kommando wäre die Virtual Machine mit dem Programm in dem EF gestartet worden. Zum Dateisystem gab es ein dazugehöriges API (*application programming interface*), sodass aus Dateien gelesen und in Dateien geschrieben werden konnte.

Diese Lösung hat sich jedoch nicht durchgesetzt. Eine Chipkarte mit Java besitzt nach der Java Card Spezifikation eine Java Virtual Machine, welche in der Kartenfertigung aktiviert und am Ende des Kartenlebenszyklus deaktiviert wird. Die Lebensdauer der Java Virtual Machine entspricht damit der Lebensdauer der Chipkarte. Die Java Virtual Machine unterbricht lediglich ihre Arbeit bei einer deaktivierten Chipkarte, sie bleibt aber aktiv. Dies ist auch ein ganz wesentlicher Unterschied zur Java Virtual Machine auf PCs, bei denen die Java Virtual Machine nur bis zum nächsten Abschalten existiert. Ein Dateisystem nach ISO/IEC 7816-4 ist nicht mehr vorgesehen, da sich dieses auch mit Objekten innerhalb eines Java Applets aufbauen lässt. Dazu existieren einige Klassen, welche den Aufbau eines ISO/IEC 7816-4-konformen Dateibaums relativ einfach ermöglichen. Der Programmcode sowie der dazugehörige Dateibaum sind dann Teil eines Applets, welches in die Chipkarte geladen wird. Es kann dort mit einer eindeutigen AID und dem SELECT-Kommando ausgewählt werden. Nach der Selektion des Applets erhält es automatisch alle APDUs zur Abarbeitung. Der Programmcode des Applets kann dann die Kommandos und deren Daten auswerten und bearbeiten sowie die entsprechenden Zugriffe auf das Dateisystem durchführen, sowie anschließend eine Antwort erzeugen.

Dieses Prinzip schafft maximale Flexibilität und Kompatibilität, da sich Anwendungen inklusive Dateibaum innerhalb eines Applets befinden. Die eigentlichen Kartenkommandos, wie READ BINARY oder MUTUAL AUTHENTICATE, sind als Programmcode innerhalb des Applets enthalten. So ist es beispielsweise möglich, gleiche Kommandos mit unterschiedlicher Codierung und differierendem Programmablauf innerhalb einer Karte unabhängig voneinander in zwei getrennten Applets zu unterstützen.

Man erkaufte sich diesen Vorteil allerdings mit einem erheblichen Speicherplatzbedarf für die jeweiligen Applets, da diese dann zwangsläufig einige redundante Daten und Programmcode teile enthalten. Um diesen eklatanten Speichermehrverbrauch in manchen Fällen etwas abzumildern, ist es möglich, Objekte eines Applets mit anderen Applets zu teilen (*object sharing*). Aus Sicherheitsgründen lässt sich dies nur von jenem Applet aus durchführen, das das Objekt auch erzeugt hat. Der Vorgang selber kann nicht mehr rückgängig gemacht werden. Das heißt, wenn der Zugriff auf ein Objekt eines Applets für ein anderes Applet freigegeben wurde, ist dies bis zum Ende des Kartenlebenszyklus der Fall.

Die einzigen Applet-unabhängigen Kommandos, die noch existieren, dienen zum sicheren Laden von Applets in die Chipkarte. Diese werden dann im EEPROM abgelegt und von der Java Virtual Machine ausgeführt. Lediglich die Übertragungsprotokolle sind einheitlich für die gesamte Chipkarte.

Java Card Framework

Um die Programmierung von Chipkarten in Java möglichst zu vereinfachen, gibt es vier Pakete, die standardisierte Schnittstellen (*application programming interface – API*) mit für Chipkarten nützlichen Funktionen zur Verfügung stellen. Drei dieser Pakete sind für alle Java Cards verpflichtend, wobei unter Umständen bestimmte Schlüssellängen oder Kryptoalgorithmen nicht vorhanden sein müssen. Das vierte Paket ist optional und mit einem „x“ für Erweiterung (*extension*) im Namen gekennzeichnet. Es kann bei Bedarf hinzugefügt werden. Es gibt noch eine Reihe weiterer applikationsspezifischer Pakete, beispielsweise für GSM-Funktionen (GSM 03.48).

Da der Programmcode „unter“ dem API in Maschinsprache für den Zielprozessor erstellt werden kann, erreicht man damit neben der einheitlichen Schnittstelle noch eine enorme Erhöhung der Ausführungsgeschwindigkeit.

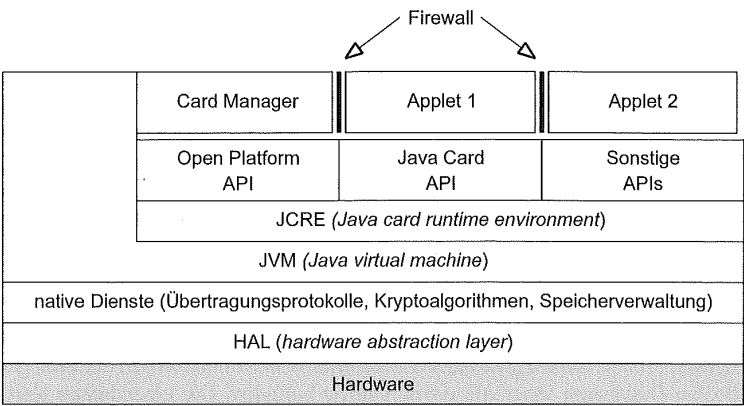


Bild 5.53 Schematische Darstellung der grundlegenden Architektur einer Java Card. Dieses Beispiel enthält einige typische APIs sowie als Anwendungsverwaltung die Open Platform Mechanismen. Die Position von Anwendungen ist weiterhin in Form zweier Applets dargestellt.

Das obligatorische Paket „java.lang“ ist die Basis für Java auf Chipkarten. In ihm sind elementare Klassen für Ausnahmen (*exceptions*) festgelegt. Eine Ergänzung dazu ist das Paket „javacard.framework“, das die Kernfunktionen für ein Java Card Applet wie elementare Klassen für Appletverwaltung, Datenaustausch mit dem Terminal und diverse Konstanten im Rahmen von ISO/IEC 7816-4 definiert. Das Paket mit kryptografischen Funktionen „javacard.security“ stellt Schnittstellen für eine Reihe von Kryptoalgorithmen zur Verfügung. Es ist aus exportrechtlichen Gründen so aufgebaut, dass mit diesem Paket die Chipkarte nicht als allgemeines Ver- und Entschlüsselungsinstrument nutzbar ist. Dazu ist dann das optionale Paket „javacardx.crypto“ notwendig, das die Schnittstellen zu den dazugehörigen Entschlüsselungsmethoden enthält.

Tabelle 5.28 Die wichtigsten Klassen des Pakets „java.lang“ des Java Card Application Programming Interface (API)

Klasse	Erklärung
java.lang.Exception	Dies ist die Klasse für die Ausnahmebehandlung für die Java Card.
java.lang.Object	Diese Klasse ist die Klasse für alle Java-Card-Klassen.
java.lang.Throwable	Diese Klasse ist die Klasse aller Ausnahmen und Fehler für die Java Card.

Tabelle 5.29 Die wichtigsten Klassen des Pakets „javacard.framework“ des Java Card Application Programming Interface (API)

Klasse	Erklärung
javacard.framework.AID	Diese Klasse kapselt den 5 bis 16 Byte langen Application Identifier (AID) nach ISO/IEC 7816-5.
javacard.framework.APDU	Diese Klasse stellt Methoden zur Verfügung, um auf APDU-Ebene Daten zwischen Chipkarte und Terminal auszutauschen.
javacard.framework.Applet	Diese Klasse legt ein Applet für eine Chipkarte fest.
javacard.framework.ISO7816	Dieses Interface kapselt verschiedene Konstanten von ISO/IEC 7816-3 und -4. Typische Beispiele dafür sind Offsetwerte zu verschiedenen Datenelementen innerhalb einer APDU und die unterschiedlichen Returncodes.
javacard.framework.JCSystem	Dies ist eine der wichtigsten Klassen. Sie besitzt beispielsweise Methoden, um Objekte persistent oder transient zu machen oder um Zugriff auf ein Objekt für mehrere Applets zuzulassen.
javacard.framework.OwnerPIN	Diese Klasse stellt Funktionen für die PIN des Kartenbesitzers bereit.
javacard.framework.PIN	Dieses Interface repräsentiert eine PIN mit dazugehörigem Fehlbedienungs-zähler und einem Gültigkeitszustand.
javacard.framework.Util	Diese Klasse enthält Methoden für den Umgang mit Feldern und Datenobjekten sowie zur Typkonvertierung.
javacardx.framework.Sharable	Dieses Interface dient dazu, sharable Objekte zu identifizieren.

Tabelle 5.30 Die wichtigsten Klassen des Pakets „javacard.security“ des Java Card Application Programming Interface (API).

Klasse	Erklärung
javacard.security.DESKey	Dieses Interface stellt den Zugriff auf DES und Triple-DES mit 2 und 3 Schlüssel zur Verfügung.
javacard.security.DSAKey	Dieses Interface stellt den Zugriff auf den DSA zur Verfügung.
javacard.security.DSAPrivateKey	Dieses Interface stellt den Zugriff auf den DSA zur Erstellung von Signaturen zur Verfügung.
javacard.security.DSAPublicKey	Dieses Interface stellt den Zugriff auf den DSA zur Prüfung von Signaturen zur Verfügung.
javacard.security.Key	Dies ist das Basisinterface für alle Schlüssel.
javacard.security.KeyBuilder	Dies ist die Klasse für die Erzeugung von Schlüssel.
javacard.security.KeyPair	Dies ist eine Container-Klasse für ein Schlüsselpaar (öffentlicher und geheimer Schlüssel).
javacard.security.MessageDigest	Dies ist die Basisklasse für Hash-Algorithmen.
javacard.security.MessageDigest	Dies ist die Basisklasse für alle Hash-Algorithmen.
javacard.security.PrivateKey	Dies ist das Interface der geheimen Schlüssel für asymmetrische Kryptoalgorithmen.
javacard.security.PublicKey	Dies ist das Interface der öffentlichen Schlüssel für asymmetrische Kryptoalgorithmen.
javacard.security.RandomData	Dies ist die Basisklasse für die Erzeugung von Zufallszahlen.
javacard.security.RSAPrivateCrtKey	Dies ist das Interface der geheimen Schlüssel für asymmetrische Kryptoalgorithmen in Verbindung mit dem chinesischem Restklassensatz (<i>chinese remainder theorem</i> – CRT).
javacard.security.RSAPrivateKey	Dieses Interface stellt den Zugriff auf den RSA zur Erstellung von Signaturen zur Verfügung.
javacard.security.RSAPublicKey	Dieses Interface stellt den Zugriff auf den RSA zur Prüfung von Signaturen zur Verfügung.
javacard.security.SecretKey	Dies ist das Interface für alle symmetrischen Schlüssel.
javacard.security.Signature	Dies ist die Basisklasse für alle Signatur-Algorithmen.

Tabelle 5.31 Die wichtigsten Klassen des Pakets „javacardx.crypto“ des Java Card Application Programming Interface (API).

Klasse	Erklärung
javacardx.crypto.KeyEncryption	Dieses Interface stellt den Zugriff auf die Schlüssel für Entschlüsselung zur Verfügung.
javacardx.crypto.Cipher	Dies ist die Basisklasse für alle Verschlüsselungsalgorithmen.

Softwareentwicklung für Java auf Chipkarten

Wie ist nun die Vorgehensweise, um ein Java-Programm für eine Chipkarte zu erstellen und zur Ausführung zu bringen? Als Erstes erstellt der Programmierer mit einem Texteditor den eigentlichen Java Source Code. Anschließend kompiliert er diesen mit einem beliebigen Java Compiler und erhält den maschinenunabhängigen Bytecode. Bis zu diesem Punkt ist der Ablauf identisch dem der Java-Programmierung für PCs.

Der Bytecode wird nun dem Java Card Converter (d. h. eigentlich dem Offcard-Teil der Java Virtual Machine) als Class-Datei übergeben, welcher Prüfungen des Formats, der Syntax, von Feldreferenzen und Ähnlichem durchführt. Sind alle Prüfungen mit positivem Ergebnis abgeschlossen, erstellt der Java Card Converter eine so genannte CAP-Datei (*card application file – CAP-file*). Diese wird je nach Anwendung und Bedarf noch mit einer digitalen Signatur versehen, um damit sicherzustellen, dass es von der Offcard-VM geprüft wurde und authentisch ist. Wäre keine überprüfbare Signatur vorhanden, könnte man mit einem manipulierten Applet die Sicherheit der Oncard-VM umgehen, denn diese kann aus Speicherplatzgründen nicht alle Prüfungen durchführen. Anschließend wird das Applet im Format der CAP-Datei in die Chipkarte geladen. Dabei kommen in der Regel

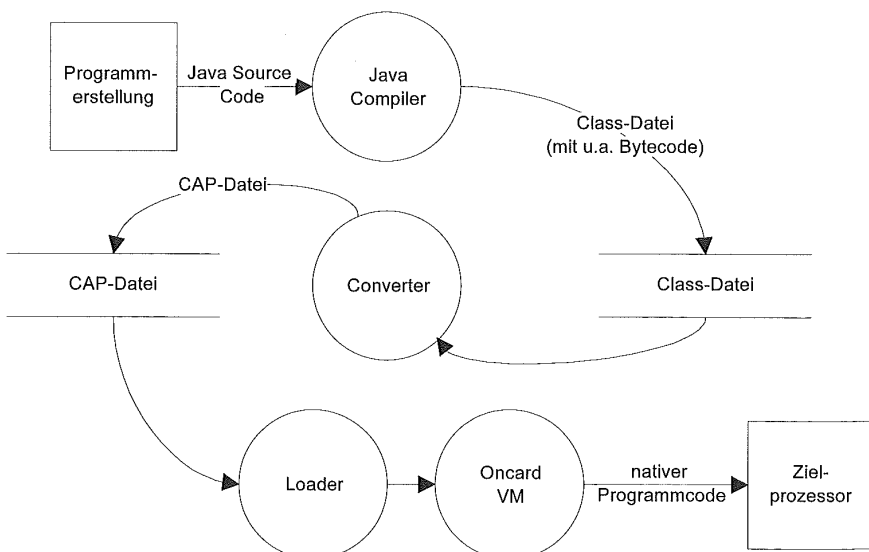


Bild 5.54 Der übliche Ablauf von der Programmentwicklung bis zur Ausführung des Programms auf der Java Virtual Machine (JVM) auf dem Chipkarten-Mikrocontroller.

die Mechanismen von OP¹⁷ zum Einsatz. Diese überprüft als Erste die – in der Regel vorhandene – digitale Signatur und übergibt das kontrollierte Applet der Oncard-VM. Was nun folgt, verhält sich wieder weitgehend analog der Programmausführung auf einer Virtual Machine auf einem PC. Die Oncard-VM prüft und interpretiert Zeile für Zeile des Bytecodes und erzeugt daraus Maschinenbefehle für den Prozessor der Chipkarte.

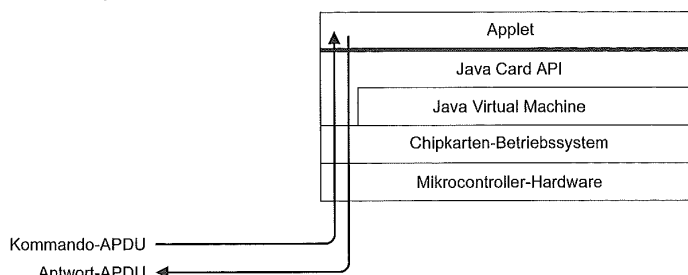


Bild 5.55 Datenfluss der Kommando-APDU zum Applet und dazugehöriger Antwort-APDU in Bezug auf das Schichtenmodell einer Java-Chipkarte.

Das obige Szenario ist natürlich in der Realität etwas aufwendiger. Der Entwickler wird hoffentlich nicht sofort nach dem Auftrag mit der Codierung in Java beginnen, sondern zuerst mit Analyse und Design die Anforderungen ermitteln und daraufhin mit der Programmierung anfangen.

Damit er während und nach der Codierung beim Debugging die Fehler auch schnell findet, benutzt er einen Simulator der Java-Chipkarte. So kann der Entwickler die Programmabarbeitung Schritt für Schritt prüfen, Variablen inspizieren und dann gegebenenfalls schnell und problemlos Korrekturen vornehmen.

Im Anschluss daran wird bei größeren und sicherheitskritischen Projekten noch eine Testsuite durchlaufen, die alle Gutfälle und die wichtigsten Fehlerfälle bei Kommando und Antwort prüft. Eventuell schließt sich daran noch eine Sourcecode-Inspektion durch unabhängige Dritte an.

Man sieht an dem Beispiel, dass Java auf Chipkarten zwar die Entwicklungszeit erheblich verkürzt und als Nebeneffekt auch die Fehlermöglichkeiten einschränkt, doch ist die reine Codierung nur ein Teil von vielen bei der Entwicklung von Anwendungen für Chipkarten. Der größte Vorteil liegt vor allem darin, dass es nun für eine Vielzahl von Entwicklern möglich geworden ist, ausführbare Programme für Chipkarten zu erstellen und nicht nur für die wenigen Softwareentwickler der Kartenhersteller. Tritt nun der Fall ein, dass ein bestimmtes Applet identisch auf einer sehr großen Anzahl von Chipkarten geladen werden muss, dann ist es bei manchen Java Card-Realisierungen auch möglich, Teile des Applets als Teil einer ROM-Maske zu definieren. Die Teile des Applets, die veränderbare Daten enthalten, bleiben jedoch im EEPROM. Man bezeichnet dies oft auch als ROMable Applet.

Um nun in Java Applets auf Chipkarten zu erstellen, sollte man neben den Besonderheiten des benutzten Betriebssystems auch einige Charakteristika der aktuellen Java Card Spezifikation berücksichtigen.¹⁸ Diese sind im Folgenden aufgeführt und kurz erläutert:

¹⁷ Siehe auch Abschnitt 5.11 „Open Platform“.

¹⁸ Einen guten Überblick über die Entwicklung von Software auf Chipkarten gibt Zhiquan Chen [Chen 00].

Ausführungsgeschwindigkeit

Der neben dem Speicherbedarf wohl größte Kritikpunkt von Java auf Chipkarten ist die geringe Ausführungsgeschwindigkeit. Es ist jedoch verhältnismäßig schwierig, faire Vergleiche zwischen Programmen in Assembler und Java zu erstellen. Die Problematik dabei liegt vor allem darin, dass man in Java nicht unbedingt einen analogen Programmablauf wie in Assembler erstellen muss, solange sich die Anwendung auf der Schnittstelle zum Terminal identisch verhält. Beispielsweise wäre ein Dateisystem bei Programmierung in Java nicht unbedingt notwendig, und auch einen Kryptoalgorithmus würde wohl niemand in Java für Chipkarten programmieren.

Generell ist bei der Programmierung auch darauf zu achten, dass man so oft wie möglich die Methoden des Java Card API benutzt, denn diese sind teilweise im nativen Programmcode des Zielprozessors erstellt, was eine erhebliche Geschwindigkeitserhöhung gegenüber dem interpretierten Java Programmcode bewirkt.

Bei der Realisierung typischer Chipkarten-Kommandos in Java mit intensiven Gebrauch der Schnittstellen zu nativen Bibliotheksroutinen kann man davon ausgehen, dass die Ausführungszeit – d. h. ohne die Übertragungszeit – etwa 50 Prozent länger ist als bei vergleichbaren Implementierungen in Assembler oder C. Bei ungünstiger Programmierung kann das Java-Programm durchaus um den Faktor 2 bis 3 langsamer sein als ein entsprechendes natives Programm.

Anwendungsselektion

Eine bestimmte Anwendung in einer Java Card ist dann selektiert, wenn das betreffende Applet mit seiner eindeutigen AID selektiert wurde. Bei dieser Selektion wird das Applet aufgerufen, damit es gegebenenfalls notwendige Initialisierungen durchführen kann. Anschließend erhält es dann automatisch alle Kommando-APDUs, die vom Terminal zur Chipkarte gesendet werden. Ist das Applet nicht selektiert, dann ist es inaktiv und auch nicht in die Datenübertragung eingebunden.

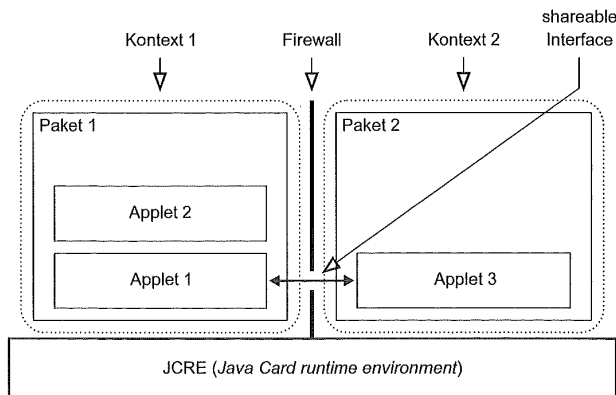


Bild 5.56 Schematische Darstellung des Zusammenhangs zweier Pakete mit zwei bzw. einem Applet zu den bei Java Card existierenden Firewalls und dazugehörigem Sicherheitskontext. Über das shareable Interface ist es den Applets 1 und 3 möglich, durch die Firewall hindurch Daten auszutauschen.

Firewalls – Anwendungstrennung

Die einzelnen Applets auf der Chipkarte sind informationstechnisch vollständig voneinander getrennt. Eine gegenseitige Beeinflussung wird durch den Security Manager der Java Virtual Machine verhindert. Bei der Java Card sind alle Applets eines Pakets automatisch innerhalb des gleichen Sicherheitskontextes. Applets innerhalb eines Kontextes können gegenseitig auf Objekte zugreifen. Fall es notwendig ist, dass Applets über Kontextgrenzen hinweg auf gemeinsame Daten (z. B. PIN) zugreifen, so stellt das JCRE Mechanismen zum sicheren Zugriff auf Objekte eines anderen Kontextes zur Verfügung. Selbstverständlich muss das Objekt, auf das von einem anderen Kontext aus zugegriffen werden soll, dies explizit erlauben.

Transaktionsintegrität – Atomare Abläufe

Ein plötzlicher Spannungsausfall während einer Sitzung darf nicht dazu führen, dass die Daten eines Applets einen undefinierten Zustand einnehmen. Dies wird bei Änderungen eines Objekts implizit durch die Virtual Machine bzw. das Betriebssystem gewährleistet. Ist es jedoch notwendig, eine unbedingte Integrität über mehrere Objekte oder Programmabläufe hinweg zu gewährleisten, dann stehen dem Entwickler eines Applets dafür spezielle Mechanismen zur Verfügung. So ist sichergestellt dass die betreffenden Objekte entweder den Ursprungszustand oder den neuen Zustand haben.

Dateisystem

Es ist für ein Applet nicht zwangsläufig notwendig, dass es ein eigenes Dateisystem besitzt. Für manche Anwendungen kann es durchaus ausreichen, dateiunabhängige Datenobjekte anzulegen, auf die dann entweder mit normierten oder selbst definierten Kommandos (*private-use*) zugegriffen werden kann. Der Vorteil an dateisystemlosen Applets ist wieder einmal die bei Chipkarten immer unterschwellig vorhandene Notwendigkeit zur Speichersparnis. Zudem können durch das objektorientierte Java Zugriffe auf Daten mit Methodenaufrufen und dazugehörigen Aufrufbedingungen für ganz spezifische Anforderungen bei bestimmten Anwendungen realisiert werden. Beispielsweise könnten dann Anwendungen oder Benutzer die Möglichkeit haben, ihre Zugriffsrechte an andere Instanzen zu vererben.

Die heute üblichen Anwendungen, sowohl im Chipkarten- als auch im PC-Bereich, zeigen jedoch, dass doch sehr oft die Daten dateiorientiert abgelegt und verwaltet werden. Diese Variante wird durch die Java Card Spezifikation nicht ausgeschlossen, jedoch leider auch nicht direkt unterstützt. Eine Dateiverwaltung, z. B. nach ISO/IEC 7816-4, muss jedoch durch eigene Klassen in Java eigens realisiert werden. Dies ist oft notwendig, da es die Grundlage dafür darstellt, um zu bisherigen Anwendungen mit ihren üblichen Dateibäumen kompatible Java-Chipkarten auszugeben.

Löschen von Objekten – Persistente und Transistente Objekte

Alle Objekte werden von sich aus als persistente Objekte im EEPROM angelegt, wenn man sie mit der Methode `new()` erzeugt. Persistenz ist die Fähigkeit eines Objekts, über die Ausführungszeit eines Prozesses hinaus zu existieren; das Gegenteil ist die Transistenz. Persistente Objekte überdauern deshalb sowohl eine Sitzung als auch plötzliche Spannungsausfälle, ohne dass Daten verloren gehen oder inkonsistent werden. Ein Objekt existiert für die Dauer, in der eine Referenz darauf verweist. Fällt die Referenz weg, dann ist das Objekt *de facto* nicht mehr vorhanden, obwohl es noch Speicher belegt. Abhilfe könnte

nur durch eine Speicherverwaltung mit einem Garbage Collector geschaffen werden, welche jedoch in der aktuellen Java Card Spezifikation noch nicht vorgesehen ist. Optional soll jedoch in der nächsten Version der Java Card Spezifikation (Version 2.2.) ein Garbage Collector vorgesehen sein.

Man kann aber persistente Objekte in transiente Objekte so umwandeln, dass sie im RAM abgelegt werden. – was jedoch nur in dieser Richtung möglich ist. Die Daten dieser Objekte gehen nach der jeweiligen Sitzung verloren und werden mit dem Standardwert neu initialisiert.

Löschen von Applets

Die Java Card Spezifikation sieht einen Mechanismus vor, Applets auf einer Chipkarte wieder zu löschen. Durch geeignete Methoden wie Defragmentierung kann der gelöschte Speicher wieder für andere Applets verwendet werden.

Kryptografische Algorithmen

Viele der heute üblichen Kryptoalgorithmen arbeiten entweder mit Änderungen und Vertauschungen auf Bit-Ebene (z. B.: DES) oder benutzen Langzahlenarithmetik (z. B.: RSA). Aufgrund der langsamen Ausführungsgeschwindigkeit und des knappen Speichers (EEP-ROM und RAM) sind Chipkarten mit Java zurzeit nicht für die Programmierung neuer Kryptoalgorithmen in Java geeignet.

Deshalb besitzen sie üblicherweise die Klasse „`javacardx.crypto`“, welche eine Schnittstelle (API) zu den in nativem Maschinencode implementierten Kryptoalgorithmen anbietet. Dies ermöglicht die Nutzung der Kryptoalgorithmen von Java-Programmen aus, ohne dabei wesentliche Geschwindigkeitseinbußen in Kauf nehmen zu müssen.

Kryptografie und Export

Chipkarten mit allgemein einsetzbaren Betriebssystemen, welche über interne Schnittstellen zur beliebigen Ver- und Entschlüsselung von Daten verfügen, sind in vielen Ländern ausfuhrgenehmigungspflichtig. Dies führt dazu, dass diese Chipkarten in manche Länder überhaupt nicht exportiert werden dürfen bzw. der Exporteur monatelang auf eine entsprechende Genehmigung des jeweiligen Ausfuhramtes warten muss.

Aus diesem Grund wurden die Klassen für Kryptografie bei Java Card in einer solchen Form aufgebaut, dass sie zwar ohne weiteres für allgemeine Datenentschlüsselung und MAC-Berechnung benutzbar sind, aber nicht zur Verschlüsselung. Für viele Anwendungen reicht dies vollkommen aus, und man kann dadurch in vielen Ländern „vereinfachte“ Ausfuhrverfahren nutzen.

Müssen jedoch in einer Anwendung Daten auch wieder entschlüsselt werden, so kann in diesem Fall der Kartenhersteller die Klassen „...`.cryptoEnc.DES3_EncKey`“ und „...`.cryptoEnc.DES_EncKey`“ einbinden, welche dann auch die Entschlüsselung ermöglichen. Aus kryptografischer Sicht könnte man sich aber durchaus einfach zu realisierende Verfahren austüfeln, um auch ohne diese beiden „Verschlüsselungsklassen“ Daten beliebig zu ver- und entschlüsseln.

Speicherplatzminimierung

Auf abschbare Zeit werden Java-Programme auf Chipkarten sehr stark von dem verfügbaren Speicher geprägt sein. Dies führt bei der Programmierung zu einigen Kompromis-

sen, die auf PCs nicht notwendig wären. Die folgende Tabelle bietet einige Ratschläge aus der Praxis, um Java-Programme optimal an die speziellen Anforderungen im Bereich von Chipkarten anzupassen.

Tabelle 5.32 Aufstellung einiger nützlicher Richtlinien zur Erstellung von Applets für Chipkarten mit Java, nach einer Idee bei [Schlumberger 97]. Der Zweck dieser Richtlinien ist es, den auf Chipkarten stark begrenzten Speicherplatz so gut wie möglich zu nutzen.

Richtlinie zur Speicherplatzminimierung	Bemerkung
Konstanten vor Variablen	Konstanten benötigen weniger Speicherplatz als Variablen und sollten deshalb bevorzugt verwendet werden.
einfache Datentypen	Die Verwendung von möglichst einfachen Datentypen spart Speicherplatz auf der Chipkarte.
Wiederverwendung von Variablen	Variablen sollten möglichst wieder verwendet werden, um möglichst wirtschaftlich mit dem begrenzten Speicherplatz auf der Chipkarte umzugehen.
lokale Variablen	Lokale Variablen sollten so sparsam wie möglich eingesetzt werden.
einfache Klassenhierarchie	Eine einfache und möglichst flache Klassenhierarchie spart Speicherplatz auf der Chipkarte.
wenige Argumente	Methoden sollten nur so viele Argumente wie unbedingt nötig benutzen.
einfache Aufrufhierarchie	Eine einfache Aufrufhierarchie der Methoden spart Speicherplatz auf der Chipkarte.
ungenutzte Variablen	Das Programm sollte nach der Erstellung immer auf ungenutzte Konstanten, Variablen, Methoden und Klassen geprüft werden, denn der Compiler könnte u. U. nicht fähig sein, diese wegzuoptimieren.

Resümee und Zukunft

Trotz des immer noch anhaltenden Hype um Java darf man nicht vergessen, dass sie sicherlich nicht Abhilfe für alle IT-Probleme der letzten und nächsten Jahre bieten wird und die in sie gesteckten Erwartungen unter Umständen nicht alle erfüllt werden. Man denke dabei nur an die vergangenen, mittlerweile aus der Mode gekommenen Sprachen wie Pascal („modular“), Lisp („KI für alle“), C („portabel“) und C++ („wiederverwendbarer Programmcode“). Diese Sprachen haben zwar die Informationstechnik um Größenordnungen weitergebracht, doch viele der prognostizierten positiven Effekte traten nie ein. Es ist jedoch eine Tatsache, dass mit Java Card eine neue Chipkarten-Ära begann, da es erstmals für Dritte in einfacher Art und Weise möglich ist, ausführbaren Programmcode in Chipkarten ablaufen zu lassen. Mittlerweile ist Java die Standard-Programmiersprache für Chipkarten-Anwendungen geworden.

Einige Entwicklungstendenzen bei Java Card sind durchaus absehbar. Die Version 2.2 der Java Card Spezifikationen wird die aktuellen Spezifikationen um einige zusätzliche sinnvolle Funktionalitäten wie Logical Channels, Garbage Collector und RMI (*remote method invocation*) ergänzen. RMI kann dazu benutzt werden, um vom Terminal aus direkt Methoden in der Chipkarte aufzurufen. Die darauf folgende Version 3 der Spezifikationen wird die Rechenleistung und Speichergrößen der neuen 32-Bit-Chipkarten-Mikrocontroller optimiert sein und deutlich mehr Möglichkeiten als die heutigen Ver-

sionen bieten. Es kann als sicher angenommen werden, dass mit der Version 3 auch eine Annäherung an das übliche Java auf PCs und dortige Funktionalitäten stattfinden wird.

Eher langfristig ist ein großflächiges Laden und Verwalten von Applets mittels Applet Management Systemen¹⁹ zu sehen, da dem eine enorme Komplexität gegenübersteht. Auf absehbare Zeit werden wohl innerhalb der üblichen Fertigungsschritte bei den Kartenherstellern Applets mit den entsprechenden individuellen Daten in Chipkarten mit Java geladen und verbleiben dann unverändert bis zum Ende der Lebensdauer.

Java Card ist zurzeit und auch auf absehbare Zeit die einzige weltweit verbreitete Technologie, um programmbasierte Anwendungen auf Chipkarten zu realisieren. Es existieren mittlerweile in den unterschiedlichen Anwendungsbereichen von Chipkarten eine Vielzahl von Spezifikationen, APIs und Ergänzungen von Java Card, die sie auf eine sehr breite Basis stellen. Zusätzlich ist die mittelfristig erreichte Kompatibilität der Java Cards unterschiedlicher Hersteller ein interessantes wirtschaftliches Argument für die Kartenherausgeber, da die Kartenhersteller austauschbar werden. Durch Java Card ist es auch möglich geworden, die gerade im IT-Bereich geforderten kurzen Entwicklungszyklen auch für Chipkarten zu realisieren. Dabei können alle bekannten und bewährten Vorgehensmodelle eingesetzt werden.²⁰ Gerade die verkürzten Entwicklungs- und Lieferzeiten kann den Chipkarten neue Märkte und Anwendungen eröffnen.

5.14.2 Multos

Das Multiapplication-Chipkarten-Betriebssystem Multos (*multiapplication operating system*) hat seine Ursprünge in der Mondex-Entwicklung für elektronische Geldbörsen. Daraus entwickelte sich in mehreren Schritten ein Chipkarten-Betriebssystem, das vor allem auf die Anforderungen des elektronischen Zahlungsverkehrs mit Chipkarten hin optimiert ist. Der Herausgeber der Spezifikationen, der Lizenzgeber und Betreiber der für Multos notwendigen Zertifizierungsdienste ist das Maosco Consortium [Maosco]. Ein Großteil der Spezifikationen von Multos sind vertraulich, weshalb hier auch nur überblickshaft auf dieses Betriebssystem eingegangen werden kann. Als interessantes Detail sind einige zentrale Betriebssystemteile von Multos nach ITSEC E6 zertifiziert, was der höchsten möglichen Stufe entspricht.

Multos entspricht den typischen ISO/IEC 7816-4 konformen Betriebssystemen und kann nachladbaren Programmcode interpretieren. Der Programmcode wird typischerweise in C entwickelt und mit einem speziellen Compiler in MEL (*Multos executable language*) übersetzt. MEL ist ein Hardware-unabhängiger Programmcode und wird von einer Stack-orientierten VM namens AAM (*application abstract machine*) ausgeführt. Von MEL aus kann die Anwendung über diverse Schnittstellen auf die Betriebssystemdienste MAOS (*multi application operating system*) zugreifen.

Um Anwendungen auf eine Multos-Chipkarte zu laden, müssen diese vorab mit relativ aufwendigen Mechanismen von lizenzierten Multos-Zertifizierungsdiensten digital signiert werden. Wie bei Zahlungsverkehrskarten üblich, sind bei Multos auch große Teile des Fertigungsprozesses im Detail festgelegt.

¹⁹ Siehe auch Abschnitt 10.5 „Phase 4 des Lebenszyklus im Detail“.

²⁰ Siehe auch Abschnitt 15.7 „Vorgehensmodelle“.

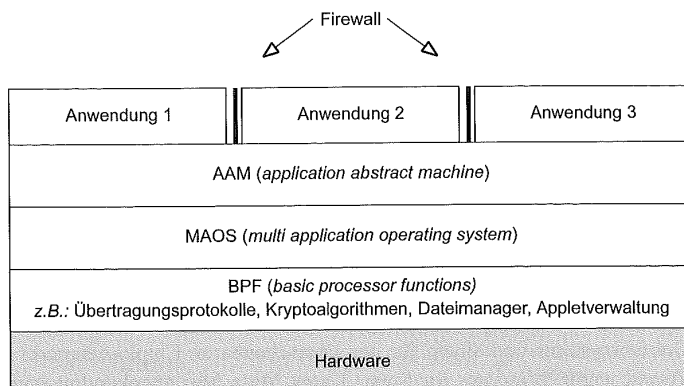


Bild 5.57 Schematische Darstellung der grundlegenden Architektur des Chipkarten-Betriebssystems Multos. Dieses Beispiel enthält aufsetzend auf Multos drei Anwendungen, erstellt in MEL.

5.14.3 Basic Card

Die deutsche Firma Zeitcontrol [Zeitcontrol] bietet seit 1996 ein Chipkarten-Betriebssystem an, das einen Interpreter für die Programmiersprache Basic enthält. Das Betriebssystem hat den Namen Basic Card und existiert in verschiedenen Versionen, Funktionalitäten und auf Hardwareplattformen unterschiedlicher Speichergrößen. Es ist neben Java Card und Multos eines der seltenen Multiapplication-Betriebssysteme, die es ermöglichen, von Dritten erstellten ausführbaren Programmcode nachzuladen.

Die Funktionsweise bei der Erstellung von nachladbaren Programmen für die Basic Card orientiert sich an den klassischen Basic-Interpretern. Der Sourcecode wird von einem Compiler in einen P-Code übersetzt, welcher von einem speziellen Ladeprogramm in einen dafür reservierten Speicher des Chipkarten-Mikrocontrollers übertragen wird. Der dort abgelegte Programmcode wird anschließend bei Bedarf vom Interpreter abgearbeitet.

Das dabei unterstützte Basic entspricht in Datentypen, Kontrollstrukturen und Funktionen dem der heute üblichen einfacheren Dialekte dieser seit Jahrzehnten existierenden Programmiersprache. Es wurde zusätzlich noch um einige Chipkarten-spezifische Funktionen erweitert, beispielsweise eine Schnittstelle zu dem Dateisystem der Chipkarte. Weiterhin werden die kontaktbehafteten Übertragungsprotokolle T=0 und T=1 sowie kontaktlose Datenübertragung nach ISO/IEC 14443 B unterstützt. Für typische Sicherheitsanwendungen können über eine Schnittstelle eine Reihe von Kryptoalgorithmen wie DES, Triple-DES, IDEA, AES, RSA mit 1024 Bit Schlüssellänge, elliptische Kurven mit 160 Bit Schlüssellänge und Hash-Algorithmus SHA-1 aufgerufen werden.

Der Programmcode ist im Vergleich zu anderen Chipkarten-Betriebssystemen mit Interpreter sehr kompakt und die Ausführungsgeschwindigkeit verhältnismäßig hoch. Beides rührt vor allem daher, dass sich prozedurales Basic einfach und schnell interpretieren lässt und keine ausgeklügelten Sicherheitsmechanismen zur Anwendungsseparierung eingesetzt werden. In bestimmten Anwendungsfällen, bei denen Programme auf der Chipkarte schnell und einfach selber erstellt werden müssen, kann die Basic Card jedoch durchaus eine Alternative zu anderen Chipkarten-Betriebssystemen darstellen.

Das Ungewöhnliche an der Basic Card ist vor allem die Tatsache, dass es sich hier um eine verhältnismäßig kleine Firma handelt, die dieses Produkt seit vielen Jahren weiterentwickelt und dahinter nicht einer der großen IT-Riesen steht, wie dies bei manchen anderen Chipkarten-Betriebssystemen mit Interpret der Fall ist.

5.14.4 Windows for Smart Cards

Im Jahr 1998 kündigte Microsoft ein Windows for Smart Cards (WSC) an. Dieses Chipkarten-Betriebssystem sollte die Bandbreite der Microsoft-Betriebssysteme erhöhen und war sicherlich auch als Alternative zu der damals noch nicht breit etablierten Java Card geplant. Nach mehreren Jahren Entwicklung, einigen Alpha-, Beta- und endgültigen Versionen des Betriebssystems und einem für den überschaubaren Chipkartenmarkt erheblichen Werbeaufwand wurde Windows for Smart Cards Mitte Mai abgekündigt und als Source Code einigen Firmen zum Kauf angeboten. Ein sonderlich großer Markterfolg stellte sich in diesem Zeitraum nie ein, da es bei der hinsichtlich Zuverlässigkeit und Sicherheit sehr anspruchsvollen Chipkartenbranche an Akzeptanz fehlte.

Windows for Smart Cards war ein für mehrere Anwendungen konzipiertes Chipkarten-Betriebssystem, das auch nachladbaren Programmcode unterstützte. Der von der VM (*virtual machine*) unterstützte Bytecode war in vielen Teilen dem Maschinencode von Prozessoren des Typs 8051 sehr ähnlich, was positiv auf die Ausführungsgeschwindigkeit niederschlug. Das Laden von Programmen in den Speicher der Chipkarte war mit den üblichen kryptografischen Mechanismen abgesichert.

Die nachgeladenen Programme konnten über mehrere APIs auf verschiedene Funktionen des Betriebssystems, wie Kryptoalgorithmen, Datenübertragung und Dateiverwaltung, zugreifen. Es gab auch ein GSM 11.14 API, um über das SIM Application Toolkit²¹ konforme Zusatzanwendungen auf SIMs zu erstellen. Die Entwicklung solcher Anwendungen konnte mit den leistungsfähigen Standardentwicklungsumgebungen für Basic und C von Microsoft durchgeführt werden. Es war lediglich notwendig, einige Bibliothekskomponenten einzubinden sowie entsprechende Compileranweisungen zu setzen.

Windows for Smart Cards war ein vollständiges Betriebssystem, es besaß also auch eine komplette Dateiverwaltung mit regelbasierten Zugriffsmechanismen. Die Dateiverwaltung orientierte sich an ISO/IEC 7816-4 und ISO/IEC 7816-9, sodass hier eine weitgehende Kompatibilität mit den heutigen Standards gegeben war. Die Dateiverwaltung hatte eine Besonderheit: sie verwendete nämlich erstmalig in einem Chipkarten-Betriebssystem eine FAT (file allocation table).²²

Bild 5.58 zeigt anhand eines Datenflussdiagramms einen Prinzipüberblick von Windows for Smart Cards. Die mit einer Entwicklungsumgebung erstellten und auf die Chipkarte geladenen Programme hatten den Namen RTE (*runtime environment*)-Anwendung. Über eine Dispatch-Tabelle wurde das Betriebssystem von einer RTE-Anwendung in Kenntnis gesetzt, sodass der entsprechende Programmcode bei Bedarf von der VM ausgeführt werden konnte. Zusätzlich war die Variante möglich, bestimmte Anwendungen oder Kommandos fest in den ROM-Code zu integrieren, was dann als non-RTE-Anwendung bezeichnet wurde.

²¹ Siehe auch Abschnitt 13.2.4 „Die SIM“.

²² Siehe auch Abschnitt 5.7 „Dateiverwaltung“.

Über einen so genannten FS (*file system*)-Builder ließen sich die für eine bestimmte Anwendung notwendigen Dateien mit allen ihren Zugriffsbedingungen erzeugen. Diese Dateien konnten dann ebenfalls auf die Chipkarte geladen werden und waren dort auch von der RTE-Anwendung zugreifbar.

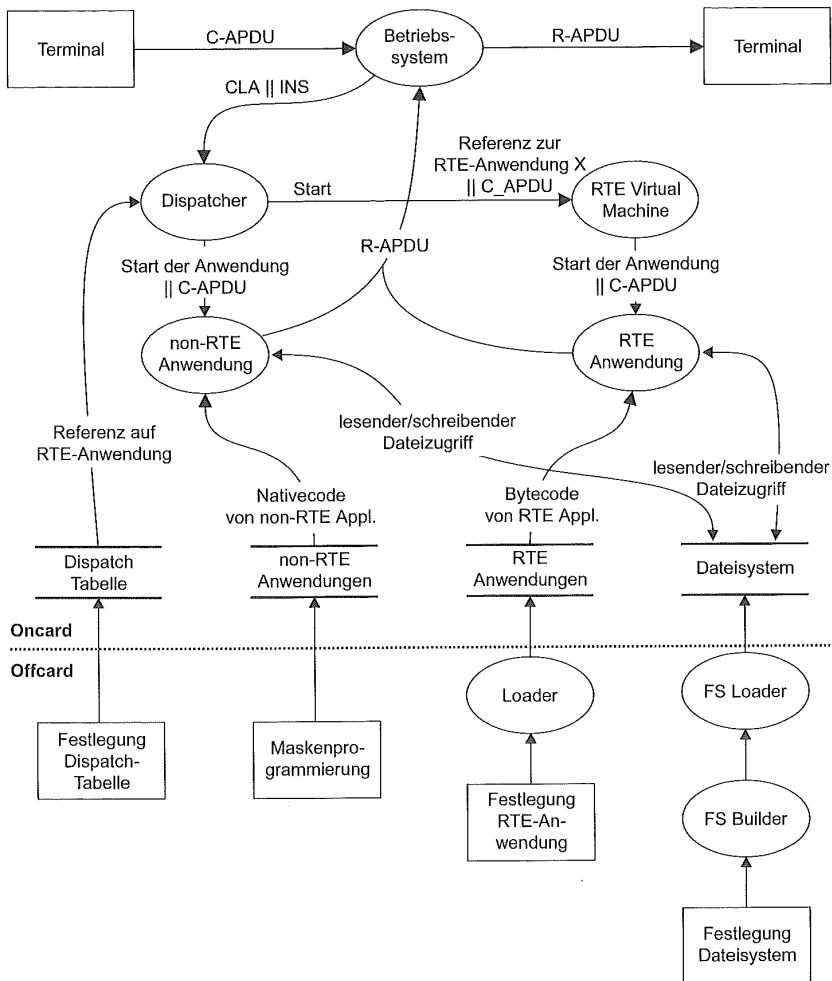


Bild 5.58 Diagramm der Datenflüsse zwischen den wesentlichen Komponenten von Windows for Smart Cards zur Demonstration der grundsätzlichen informationstechnischen Zusammenhänge. Das Diagramm wurde mithilfe der veröffentlichten Beschreibungen erstellt und erhebt keinen Anspruch auf Vollständigkeit.

5.14.5 Linux

Das Open Source Betriebssystem Linux verändert seit Ende der 90er-Jahre große Teile der Software Branche. Der Fokus von Linux waren bisher im Vergleich zu Chipkarten leistungsfähige Rechner im PC-Bereich. Es gibt jedoch seit einiger Zeit vermehrt Bestrebungen, Linux auch im Umfeld von typischen Mikrocontroller-Anwendungen zu etablieren. Voraussetzungen dafür sind bisher allerdings Rechenleistungen, die nur von typischen 32-Bit-Prozessoren erbracht werden können und Speicherbedürfnisse im Bereich einiger hundert kByte ROM und einiger zehn kByte RAM. Diese Anforderungen können von heutigen Chipkarten-Mikrocontrollern noch nicht erbracht werden. Es ist jedoch durchaus vorstellbar, dass die bisherigen Linux-Versionen seitens ihrer Hardwareanforderungen noch weiter reduziert werden können. Gleichzeitig steigt die Leistungsfähigkeit der Chipkarten-Mikrocontroller von Generation zu Generation, sodass es in nicht allzu ferner Zeit möglich sein wird, dass Linux für Chipkarten verfügbar ist.

Neben Linux kann in diesem Zusammenhang selbstverständlich auch ein anderes Open-Source-Betriebssystem auf Chipkarten in Erscheinung treten. Wichtig ist dabei nur, dass es lizenzfrei verwendet werden darf, denn dies ist aufgrund der großen Stückzahlen und damit verbundenen hohen Lizenzkosten eine der größten Barrieren beim Einsatz von Standard-Betriebssystemen auf Chipkarten.

5.15 Chipkarten-Betriebssystem „Small-OS“

Die Eigenschaften und grundlegenden Funktionsweisen von Chipkarten-Betriebssystemen sind vorangehend im Überblick beschrieben worden. Für diejenigen Leser, die jedoch tiefer in die Thematik eintauchen möchten, ist auf den folgenden Seiten ein vollständiges Betriebssystem aufgezeigt. Es demonstriert im Detail die Zusammenhänge innerhalb eines klassischen Chipkarten-Betriebssystems nach ISO/IEC 7816-4 bzw. GSM 11.11.

Das im Folgenden beschriebene Chipkarten-Betriebssystem hat den Namen „Small-OS“, der daher rührt, dass es nur sehr wenig Speicher benötigt und auch auf einer nicht sehr leistungsfähigen Hardwareplattform lauffähig wäre. Geschrieben ist es in einem BASIC-ähnlichen Pseudocode, der in dieser Form nicht direkt kompiliert werden kann, da einige Anweisungen nicht bis ins letzte Bit auscodiert und nur verbal ausgeführt sind. Wir wollen hier nicht seitenweise unverständlichen und langweilig zu lesenden echten Programmcode in einer Sprache wie C oder C++ präsentieren, sondern ein anschauliches und einführendes Beispiel aufzeigen.

Die Verständlichkeit von Pseudocode überwiegt den Vorteil der sofortigen Kompilierung und Ausführbarkeit einer echten Programmiersprache bei weitem. Dadurch verliert man sich nicht in Details der Implementation, sondern kann sich ganz und gar auf die grundlegenden Abläufe konzentrieren. Die folgende Darstellung ist eine plattformfreie Implementierung, die nicht auf eine bestimmte Hardware zurechtgeschnitten ist. Es existiert aber eine Realisation des Small-OS, allerdings nicht in Assembler auf einem Chipkarten Mikrocontroller, sondern als Simulation in dem Programm „The Smart Card Simulator“. Es ist kostenlos als Sourcecode unter GPL via Internet erhältlich [Rankl].²³

²³ Aus zeitlichen Gründen und wegen mangelnder Nachfrage war es leider nicht mehr möglich, „The Smart Card Simulator“ weiterzuentwickeln, weshalb der verfügbare Entwicklungsstand unvollständig ist und auch in vielen Punkten nicht mehr dem aktuellen Stand der Normung entspricht.

Tabelle 5.33 Die Funktionen von Small-OS im Überblick

Name des Betriebssystems	Small-OS
typisches Anwendungsfeld	Multiapplikation ohne eigenen Programmcode
Hardwarevoraussetzungen	ROM: ≈ 8 kByte, EEPROM: ≈ 1 kByte, RAM: ≈ 128 Byte
Befehlssatz	Profil N nach ISO/IEC 7816-4 mit Erweiterungen Kommandos: SELECT FILE, READ BINARY, UPDATE BINARY, READ RECORD, UPDATE RECORD, VERIFY, INTERNAL AUTHENTICATE
Datenübertragung	Übertragungsprotokoll T = 1 Teiler (CRCF) fest auf 372 eingestellt (Standard nach ISO/IEC 7816-3) PTS nicht unterstützt Secure Messaging nicht unterstützt
Dateisystem	eine DF-Ebene Working EF Strukturen: transparent und linear fixed Internal EF Strukturen: linear variable (für PIN und Schlüssel) ein EF Key pro Verzeichnis (d. h. pro MF/DF) zulässig kein dynamisches Dateisystem (d. h. kein Löschen und Anlegen von Dateien, keine Freispeicherverwaltung) maximale Größe eines transparenten EFs: 255 Byte
Zustandsautomat	unabhängige Sicherheitszustände für MF und DF Anzahl der Sicherheitszustände 256 (0 . . . 255) Initialzustand = 0 nur ein erlaubter Eingangszustand für die Benutzung einer PIN/eines Schlüssels nur ein erlaubter Eingangszustand für Dateizugriff (d. h. keine „>“ oder „≥“ Vergleiche möglich) keine ebenenübergreifenden Schlüsselzugriffe (d. h. das EF Key wird immer über das aktuell selektierte Verzeichnis (MF oder DF) ausgewählt) PIN-Adressierung: maximal 2 PINs; (Ref. Nr. 1 und 2) möglich Schlüssel-Adressierung: maximal 31 Schlüssel (Ref. Nr. 1 . . . 31) möglich die Fehlbedienungszähler für PIN und Schlüssel lassen eine maximale Anzahl von 15 Fehlversuchen zu
kryptografischer Algorithmus	DES
Programmcode	Erstellen und Laden durch Externe nicht möglich

Programmierung in Pseudocode

Zum Programmierstil und zur Programmierung von Small-OS seien hier noch einige Anmerkungen gemacht. Der an BASIC angelehnte Pseudocode stellt dem Prinzip nach eine semiformale Beschreibung des Chipkarten-Betriebssystems Small-OS dar. Bei Software-Evaluierungen nach ITSEC erstellt man oft eine ähnliche Charakterisierung des Betriebssystems, die dann als Grundlage von Evaluierung und Sourcecode-Prüfung benutzt wird. Deshalb bietet dieser in Tabellenform aufgeführte und ausführlich dokumentierte Pseudocode auch ein gutes Beispiel, wie in der Praxis formalisierte Abläufe innerhalb von Chipkarten-Betriebssystemen dargestellt werden. Ähnliche Formen der Veranschaulichung finden sich beispielsweise in der Normenreihe EN 1546, in der die internen Abläufe von Chipkarten bei elektronischen Geldbörsen semiformal beschrieben sind.

Die einzelnen Funktionalitäten des Pseudocodes sind am Anfang des Buches im Detail beschrieben.²⁴ Der Programmcode als solcher orientiert sich an den üblichen BASIC-Dialekten mit objektorientierter Erweiterung. Es wurden ausschließlich allgemein verständliche Konstrukte benutzt. Alle Labels, Konstanten und Anweisungen sind einheitlich in Englisch, um die in den meisten Fällen unglückselige Eindeutschung von State-of-the-art-Begriffen zu vermeiden. Zahlenwerte sind grundsätzlich in hexadezimaler Schreibweise und in der Notation von ISO (z. B.: '42') angegeben. Wo es jedoch für das Verständnis erforderlich ist, wurde die dezimale oder binäre Schreibweise in einer an die ISO angelehnten Notation gewählt. So sind etwa alle zählbaren Werte, wie z. B. Längenangaben, in dezimaler Darstellung aufgeführt.

In Assembler oder C würde niemand in dieser Form ein Chipkarten-Betriebssystem programmieren, da es viel zu umständlich wäre. Eines der Designziele bei Small-OS war, in möglichst verständlicher und gut kommentierter Form ein einfaches, aber dennoch leistungsfähiges Chipkarten-Betriebssystem zu erstellen. Auf eine Minimierung von Programmlaufzeiten, Programmcode, RAM-Bedarf oder Stacktiefe wurde bewusst verzichtet, da dies ganz erheblich die Lesbarkeit beeinträchtigt hätte. Es ist beispielsweise bei der echten Programmierung von Chipkarten-Betriebssystemen manchmal üblich, seltene Unterprogrammaufrufe nicht mit einem CALL-Befehl, sondern mit JUMP durchzuführen, um somit 2 Byte auf dem kostbaren Programmstack zu sparen. Ein vor dem Programmaufruf mit JUMP gesetztes Flag wird dann nach der Abarbeitung des Unterprogramms zur Ermittlung der Rücksprungadresse benutzt. Diese Art von „Optimierung“ wird man im Small-OS aus oben genannten Gründen nicht antreffen. Optimierte wurde nur in Hinblick auf Lesbarkeit und Verständlichkeit. Die aus den genannten Gründen notwendigen Abweichungen von Small-OS zu echten Chipkarten-Betriebssystemen sind an den betreffenden Stellen im Text oder im kommentierten Pseudocode gekennzeichnet.

Die größten Teile eines Chipkarten-Betriebssystems befinden sich aufgrund des benötigten Speicherplatzmangels von Chipkarten-Mikrocontrollern im ROM. Sie sind damit nach der Chip-Produktion nicht mehr veränderbar. Da Software, ausgenommen triviale Miniprogramme, nie fehlerfrei, sondern nur fehlerarm erstellt werden kann, hätte ein Programmierfehler im ROM-Programmcode schwerwiegende Auswirkungen. Um hier die Möglichkeit zu schaffen, mit einem Bug Fix solche Fehler zu patchen, sieht man an kritischen Stellen im ROM Aussprünge ins EEPROM vor. Das Prinzip ist in der Informationstechnik uralte und nicht chipkartenspezifisch. So finden beispielsweise im MacOS oder OS/9 beinahe identische Mechanismen Anwendung. Man verwendet dazu so genannte „Handles“, d. h. Stellen im EEPROM, die vom Programmcode aufgerufen werden und normalerweise nur die Anweisung des sofortigen Rücksprungs erhalten („RETURN“). Ist ein Bug Fix des ROM-Programmcodes notwendig, dann wird nicht sofort zurückgesprungen, sondern im EEPROM wird ein korrigierter Programmteil eingefügt. In Small-OS wurde aus Gründen der Vereinfachung und Verständlichkeit auf alle diese Mechanismen verzichtet.

Durch die relativ detaillierte Beschreibung des Chipkarten-Betriebssystems ergibt sich die interessante Gelegenheit, einige typische Angriffe direkt im Pseudocode zu verfolgen. An geeigneter Stelle wird jeweils im Detail auf mögliche Attacken und deren Abwehr auf Betriebssystemebene eingegangen. So kann man beispielsweise den mittlerweile schon klassischen Angriff auf die PIN mittels Zeitvergleich im Pseudocode in allen Einzelheiten

²⁴ Siehe auch Abschnitt „Programmcode“.

verfolgen. Eine umfassende Aufstellung von typischen Angriffen und deren Abwehr befindet sich in Kapitel 8 – Sicherheitstechnik.

Designkriterien

Obige Punkte führen zu den folgenden Designkriterien, welche bei der Konzeption und Programmierung von Small-OS Pate gestanden haben. Small-OS sollte ein einfaches Chipkarten-Betriebssystem werden, das ähnlich wie die großen Vorbilder wenig Programmcode erfordert und dessen Komplexität gering ist. Somit ist es in seiner Struktur eingängig und leicht begreifbar. Es ist streng modular aufgebaut und könnte deshalb auch mit vernünftigem Aufwand ohne weiteres um zusätzliche Kommandos erweitert werden. Dateisystem und unterstützte Kommandos halten sich ohne Einschränkung an den weltweiten Standard ISO/IEC 7816-4. Als Option wurde das Profil N nach ISO/IEC 7816-4 mit einigen in der Chipkartenwelt üblichen Erweiterungen gewählt.

Tabelle 5.34 Die nach Priorität geordneten Designkriterien von Small-OS

Priorität	Kriterium	Begründung
1	kompatibel zu ISO/IEC 7816-4	diese Norm ist der weltweite Industriestandard bei Chipkarten-Betriebssystemen
2	robust	hohe Zuverlässigkeit große Fehlertoleranz
3	niedrige Komplexität	hohe Zuverlässigkeit leichte Verständlichkeit der Funktionsweise
4	modularer Aufbau	hohe Zuverlässigkeit große Fehlertoleranz einfache Erweiterbarkeit
5	geringer Speicherbedarf	ein absolutes Muss für Chipkarten-Betriebssysteme
6	Multiapplication-Betriebssystem	mittlerweile übliche Funktionalität
7	kein nachladbarer Programmcode	hohe Zuverlässigkeit geringe Komplexität
8	ähnlich echten Chipkarten-Betriebssystemen	Small-OS ist ein Lehrbeispiel mit Bezug zur realen Welt

Small-OS ist deshalb für einen Einsatz gedacht, bei dem nach Ausgabe der Karten keine Anwendungen nachgeladen werden müssen. Es können jedoch im Rahmen des zur Verfügung stehenden Speicherplatzes mehrere Anwendungen unabhängig voneinander auf der Chipkarte betrieben werden; deshalb ist Small-OS ein Multiapplication-Betriebssystem. Es ist aber nicht möglich, ausführbaren Programmcode nachträglich auf die Karte zu laden und dort auszuführen. Zusammengefasst ist Small-OS damit vergleichbar mit den ersten allgemeinen Chipkarten-Betriebssystemen, wie sie beispielsweise immer noch in Varianten bei GSM eingesetzt werden.

Die ISO/IEC 7816-4 ist eine Norm, die unter anderem ein Basisdateisystem und einige grundlegende Kommandos für Chipkarten beschreibt. Damit charakterisiert sie vor allem die Schnittstelle zur Chipkarte und weniger den internen Aufbau des Betriebssystems. Es gibt des weiteren noch zahlreiche Optionen und leider auch einige auslegungsfähige Passagen in dieser Norm. Die damit geschaffenen Variationsmöglichkeiten müssen in der Pra-

xis durch Spezifikationen, wie z. B. die GSM 11.11, stark reduziert werden, um eine Kompatibilität zwischen unterschiedlichen Realisationen sicherzustellen.

Die Feststellung „kompatibel zu ISO/IEC 7816-4“ bedeutet deshalb bei realen Chipkarten-Betriebssystemen noch lange nicht, dass sie sich in allen Punkten gleich verhalten. Dazu bräuhete es eine detaillierte Spezifikation, welche wiederum in Teilen eine individuelle

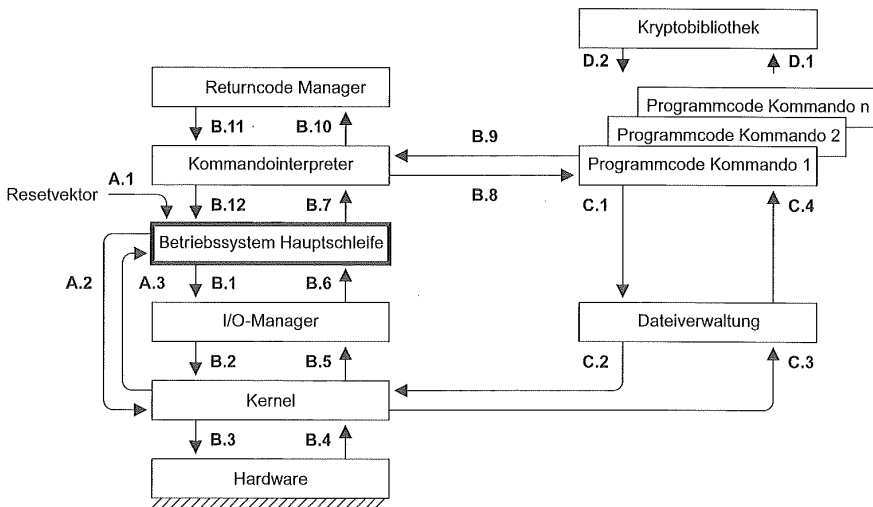


Bild 5.59 Das Schichtenmodell von Small-OS mit der daraus resultierenden Aufrufsystematik. Aufrufe mit der Kennzeichnung „A“ werden beim Start des Betriebssystems benutzt, „B“-Aufrufe beim Ausführen von Kommandos und Aufrufe der Kategorie „C“ bei Bedarf innerhalb eines Kommandos bei Zugriffen auf das Dateisystem. Die Aufrufe mit der Bezeichnung „D“ finden bei Benutzung einer kryptografischen Funktion statt. Die Zahlen bezeichnen die Reihenfolge bei Programmaufrufen.

Interpretation von ISO/IEC 7816-4 wäre. Meist ist deshalb in der Praxis einzig und allein der Gutfall bei einem Kommando identisch, im Fehlerfall gibt es diverse Unterschiede. Bei Small-OS sind Interpretationen der ISO/IEC 7816-4 im Pseudocode in der Regel als solche gekennzeichnet. Im Rahmen der Normauslegung ist Small-OS ein echtes ISO/IEC 7816-4-konformes Betriebssystem, das so weit wie möglich den in der Chipkartenindustrie üblichen Interpretationen von ISO/IEC 7816-4 entspricht.

Größere Unterschiede der Betriebssysteme untereinander sind oftmals im Bereich der Returncodes zu finden. Da in der ISO/IEC 7816-4 die einzelnen Returncodes in Verwendung und Priorität nicht im Detail beschrieben sind, müssen zwangsläufig Annahmen gemacht werden. Small-OS hat jedoch gegenüber allen anderen Betriebssystemen den Vorteil, dass der Programmcode offen gelegt ist und damit im Detail nachvollzogen werden kann, an welcher Stelle der jeweilige Returncode erzeugt wird.

Würde man Small-OS auf einen Chipkarten-Mikrocontroller implementieren, wären in Assembler für einen 8051 Prozessor etwa 5 bis 6 kByte ROM, 128 Byte RAM und je nach Anzahl der für die Anwendung notwendigen Dateien mindestens 1 kByte EEPROM erforder-

derlich. Als Rahmenbedingung sei hier noch die alleinige Verwendung des Übertragungsprotokolls T = 1 und DES als Kryptoalgorithmus vorausgesetzt. Benötigen bestimmte Anwendungen mehr Speicherplatz, dann könnte ohne Probleme ein Mikrocontroller mit größerem EEPROM verwendet werden. Auswirkungen oder Änderungen im Chipkarten-Betriebssystem wären nicht die Folge.

Zugriff auf Dateien

Das in der ISO/IEC 7816-4 beschriebene Dateisystem lässt, was Zugriffsbedingungen und Schlüsselmanagement angeht, eine enorme Vielfalt zu. Bei Small-OS wurde deshalb eine in der Praxis bei Multiapplikations-Betriebssystemen häufig benutzte Lösung gewählt. EFs des Typs „working“ besitzen im Dateiheder jeweils für die unterschiedlichen Zugriffskommandos (z. B.: READ) einen vorgegebenen Zustand (z. B.: „... .AccessCondition.Read“). Dieser wird durch eine positive Ganzzahl beschrieben. Damit besitzt jedes EF in seinem Dateiheder unabhängig für Lese- und Schreibzugriff einen Zustandswert, der im aktuellen Verzeichnis erreicht sein muss, um das Kommando auszuführen. Der Zustand „0“ stellt den Grundzustand („idle“) dar, ist also derjenige, bei dem alle Zugriffe erlaubt sind.

Dem MF und dem aktuell selektierten DF sind jeweils eine Zustandsvariable („SecurityState.MF“, „SecurityState.DF“) zugeordnet, die durch sicherheitsrelevante Kommandos (VERIFY, INTERNAL AUTHENTICATE) verändert werden können. Beim Zugriff auf ein bestimmtes EF wird nun der verlangte Zustand im Dateiheder des EFs mit dem aktuellen Zustand im Verzeichnis auf Gleichheit verglichen. Sind geforderter und aktueller Zustand gleich, darf auf die Datei lesend bzw. schreibend zugegriffen werden.

Echte Chipkarten-Betriebssysteme lassen hier oftmals eine große Variabilität von Kleiner-, Größer-, Größer-Gleich- und Ungleich-Vergleichen zu. Auch können im Dateiheder oft mehrere mögliche Zugriffszustände unabhängig voneinander definiert werden. Dies macht das ohnehin nicht gerade anschauliche Verfahren nochmals komplizierter und wird deshalb hier nicht verwendet. Prinzipiell könnte Small-OS aber ohne strukturelle Änderungen dahingehend erweitert werden.

Zugriffe auf interne Geheimnisse (PINs und Schlüssel)

Alle PINs und die Schlüssel sind in speziellen EFs des Typs „internal“ abgelegt. Diese EFs werden hier EF Key genannt. Dateien dieses Typs können nur vom Betriebssystem selbst gelesen oder geschrieben werden. Eine Selektion oder ein Zugriff von außen ist nicht möglich. Es existieren nicht einmal im Ansatz Mechanismen, um auf solche EFs zuzugreifen. Dies ist ein Teil der Sicherheitsphilosophie des Chipkarten-Betriebssystems Small-OS.

Ein EF Key kann pro Verzeichnis nur einmal angelegt werden und hat automatisch die Struktur „linear variable“, um unterschiedlich lange PINs und Schlüssel speicherplatzminimiert aufzunehmen. Jeder Record in einem EF Key enthält entweder eine PIN oder einen Schlüssel mit einer im EF Key eindeutigen Nummer zur Adressierung („... .KeyNo“). Zu jedem Geheimnis (d. h. PIN bzw. Schlüssel) ist jeweils ein Zustand gespeichert, der notwendig ist, um es in einem Kommando (VERIFY, INTERNAL AUTHENTICATE) zu benutzen („... .EntryState“). Je nach Ergebnis des Kommandos (z. B. PIN-Vergleich erfolgreich oder nicht erfolgreich) wird daraufhin in dem dem EF Key zugehörigen Verzeichnis der Ergebniszustand gesetzt („... .ResultState.OK“, „... .ResultState.NOK“). Zusätzlich ist jedem Geheimnis ein Fehlbedienungs-zähler („... .RCntr“)

zugeordnet, der im Falle eines nicht erfolgreichen Ausgangs bis zu seinem Maximalwert erhöht wird. Hat der Fehlbedienungs­zähler seinen Maximalwert („...RCntrMaxValue“) erreicht, kann das Geheimnis nicht mehr benutzt werden.

Manche Chipkarten-Betriebssysteme lassen so genannte Ebenen-übergreifende Schlüsselzugriffe zu. Damit ist es möglich, innerhalb eines DFs auf Schlüssel in dem in der Verzeichnishierarchie nächsthöheren Verzeichnis zuzugreifen. Die dazu erforderlichen Mechanismen wurden in Small-OS nicht integriert, da der Umfang des dafür notwendigen Programmcodes den Funktionalitätsgewinn nicht gerechtfertigt hätte. Aus dem gleichen Grund wurden keine Schlüsselzugriffe über Alias-Mechanismen implementiert.

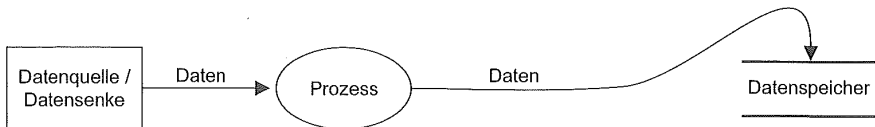


Bild 5.60 Die verwendeten Symbole für die Datenflussanalyse von Small-OS. Die Datenquelle/-senke wird auch als Terminator bezeichnet und kennzeichnet ein Objekt, das sich außerhalb des betrachteten Systems befindet und mit dem System Daten austauscht. Ein Prozess ist innerhalb des betrachteten Systems, bearbeitet eingehende Datenflüsse und erzeugt ausgehende Datenflüsse. Der Speicher ist ein Depot für Daten, die gelesen und geschrieben werden können. Eine detaillierte Einführung in die Systemanalyse findet sich bei Robertson [Robertson 96].

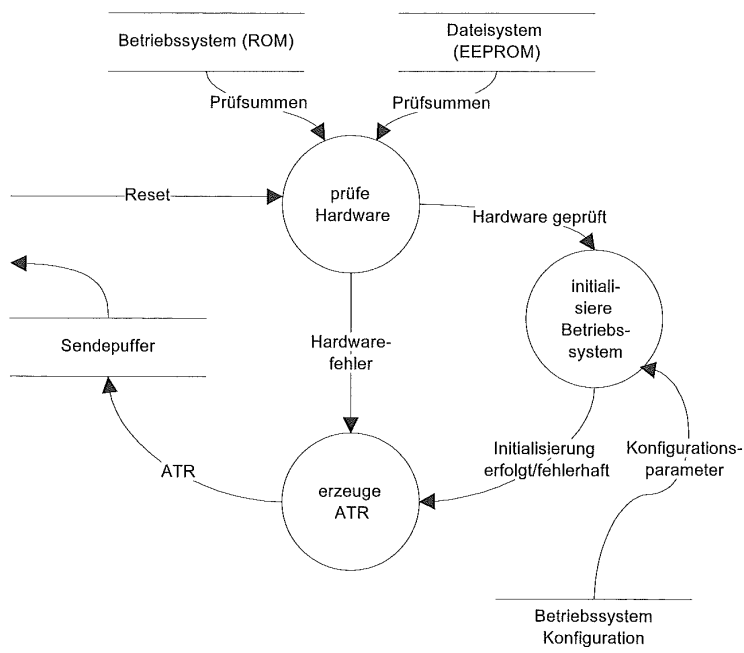


Bild 5.61 Small-OS, Datenflussanalyse bei den Start- und Testprozessen des Betriebssystems.

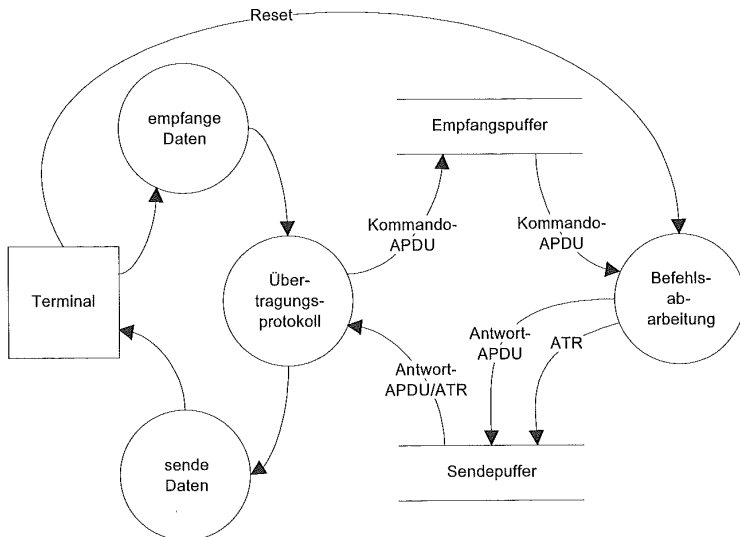


Bild 5.62 Small-OS: Datenflussanalyse für Reset und Datenübertragung.

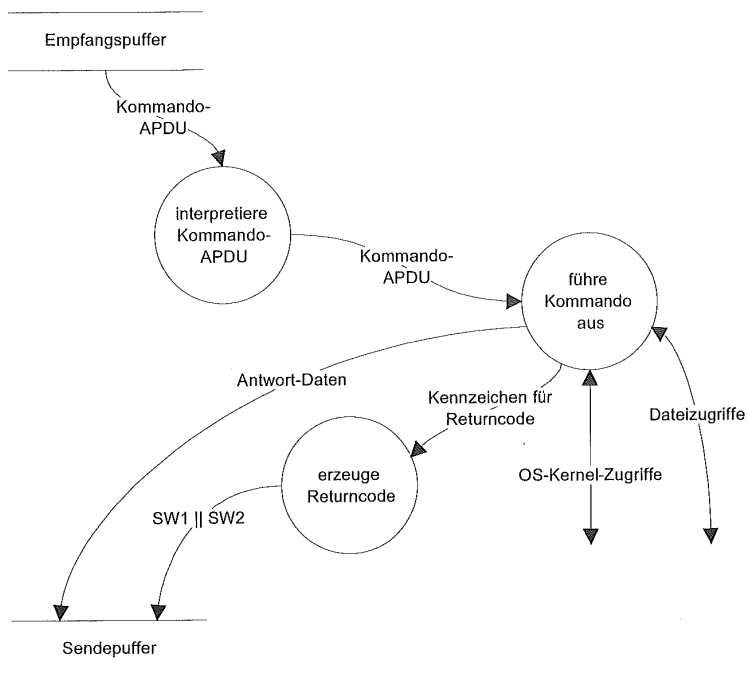


Bild 5.63 Small-OS, Datenflussanalyse innerhalb des Prozesses der Kommandoabarbeitung.

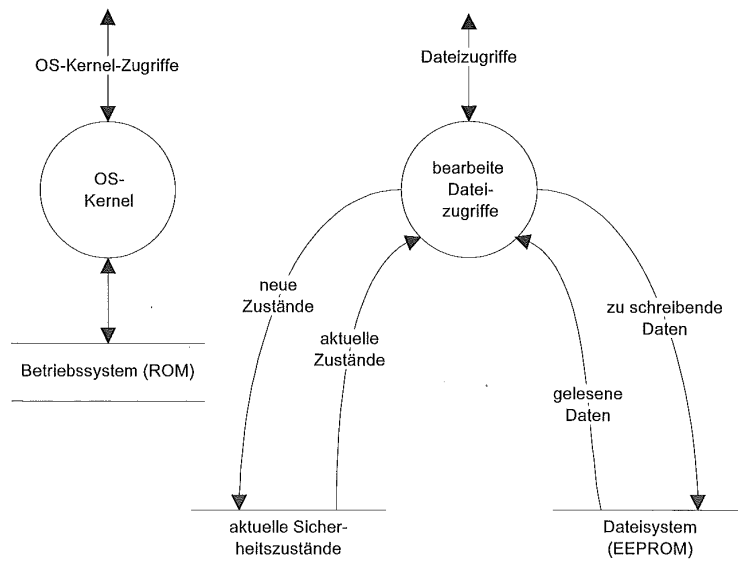


Bild 5.64 Small-OS, Datenflussanalyse innerhalb der Prozesse der Datei- und OS-Kernel-Zugriffe.

Konstanten von Small-OS

Grundsätzlich wurden im Pseudocode so weit als sinnvoll Konstanten für alle numerischen und auch nicht-numerischen Werte verwendet. Damit steigt die Lesbarkeit erheblich, und es ist schlicht und einfach guter Programmierstil. Alle Konstanten sind mit dem Präfix „C_“ gekennzeichnet. Der Wert dieser Konstanten ist in der Regel Hardware- oder Implementationsabhängig und hier nicht weiter festgelegt. Die Konstanten für die zwei Byte langen Returncodes haben einheitlich den Präfix „C_RC_“. In Tabelle 5.22 ist der dazugehörige Returncode aufgeführt. In der Praxis sind die Konstanten eines Betriebssystems meist unveränderlich im ROM abgelegt.

Variablen von Small-OS

Die Variablen von Small-OS lassen sich grundsätzlich in RAM und EEPROM-Variablen aufteilen. Die RAM-Variablen werden nach jedem Reset der Chipkarte neu initialisiert und können ihren Wert nur während einer Sitzung behalten. Dafür können in RAM-Variablen ohne zeitliche Verzögerung Daten geschrieben werden, und die Anzahl der Schreibzyklen ist unbegrenzt. EEPROM-Variablen hingegen werden typischerweise vor allem für die Realisation der Dateiverwaltung verwendet, da Dateninhalte und Zugriffsbedingungen auch zwischen Sitzungen Bestand haben müssen.

Variablen von Small-OS im RAM

Der Bereich für Sende- und Empfangspuffer belegt den größten Teil der RAM-Variablen. Alle Datenelemente einer APDU können über eigens spezifizierte Variablen adressiert werden. Um mit dem spärlichen Platz im RAM hauszuhalten, werden in der Praxis

manchmal Teile von Sende- und Empfangspuffer überlappend aufgebaut. Manche Kommandos müssen dann aber eine Ausnahmebehandlung vom Betriebssystem erfahren, da in den Sendepuffer erst dann geschrieben werden kann, wenn der Empfangspuffer vollständig abgearbeitet ist. Aus Verständnisgründen wurde keine Speicherplatzminimierung angewandt, und Sende- und Empfangspuffer sind deshalb vollständig voneinander getrennt.

Die zweite große Gruppe der RAM-Variablen umfasst die Verwaltung des Dateibaums. Dazu sind mehrere Zeiger vorgesehen, die auf das aktuelle Verzeichnis („... .Ptr.CurrentDF“), die aktuelle Datei („... .Ptr.CurrentWEF“) und auf die aktuell gültige Datei mit den Schlüsseln („... .Ptr.CurrentIEF.Key“) zeigen. Bei Record-orientierten EFs (linear fixed EF) wird der aktuell ausgewählte Record ebenfalls über einen Zeiger adressiert („... .Ptr.CurrentRecord“). Alle Zeiger sind durch ein vorangestelltes „Ptr.“ gekennzeichnet und sind, wenn sie nicht benutzt werden dürfen, ausdrücklich auf den Wert „C_InvalidPointer“ gesetzt.

Zur Kennzeichnung des Sicherheitszustands im MF bzw. im aktuell selektierten DF gibt es die beiden Variablen „SecurityState.MF“ und „SecurityState.DF“. In ihnen ist der für das jeweilige Verzeichnis erreichte Sicherheitszustand als positive Ganzzahl gespeichert. Der Wert „0“ gibt dabei an, dass kein Sicherheitszustand erreicht ist, und wird beim Initialisieren von Small-OS nach einem Reset automatisch gesetzt.

Weiteren Speicher im RAM würden Programmstack, kryptografischer Algorithmus DES und Arbeitsregister benötigen. Hier wird aber davon ausgegangen, dass dies implizit der Fall ist, und es sind deshalb keine speziellen Variablen dafür ausgewiesen.

Variablen von Small-OS im EEPROM

Der Dateibaum in der Chipkarte wurde aus Vereinfachungsgründen in diesem Lehrbeispiel als mehrdimensionales Feld realisiert. Dies wäre für eine Realisierung in einem echten Chipkarten-Betriebssystem viel zu speicheraufwendig. Dort kommen üblicherweise als grundlegende Datenstrukturen für die Dateiverwaltung einseitig verkettete Listen zum Einsatz. Diese Listenelemente können mithilfe einer TLV-Codierung in ihrer Länge variabel gehalten werden, sodass der Speicherverbrauch für die Dateiverwaltung minimiert wird, weil dann nur die notwendigen Datenelemente im Speicher gehalten werden müssen. Übliche Dateiholder für DFs oder EFs mit allen für diese Datei notwendigen Informationen besitzen eine Größe zwischen 20 und 40 Byte. Bei Realisation in einer Hochsprache in Hardware-Umgebungen mit geringen Speicherplatzbeschränkungen kann jedoch die in Small-OS verwendete Dateistruktur mit mehrdimensionalen Feldern durchaus die einfachere und effizientere Lösung sein. Beispielsweise wird in dem Programm „The Smart Card Simulator“ ein nahezu identischer Aufbau benutzt.

Der Anfangspunkt für die Dateiverwaltung ist immer „DF[...]“. Selbst das MF ist damit nur eine besondere Form des DF. Es ist nicht vorgesehen, dass es DFs unter anderen DFs gibt. Die EFs mit ihren Dateninhalten und Eigenschaften werden in logischer Konsequenz zu Obigem über „DF[...]EF[...]“ adressiert. Dabei gibt es je nach EF-Typ (*working*, *internal*) und EF-Struktur (*transparent*, *linear fixed*, *linear variable*) unterschiedliche Variablen, die je nach Datei-Typ und Datei-Struktur die charakteristischen Eigenschaften des EFs enthalten.

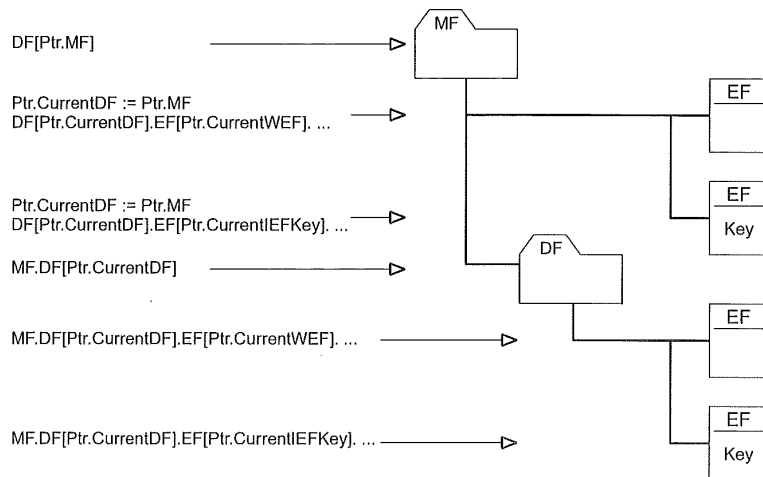


Bild 5.65 Beispiele für die Adressierung von Verzeichnissen (MF, DF) und Dateien (EF) innerhalb der von Small-OS zur Verfügung gestellten Datenstrukturen.

Tabelle 5.35 Tabelle der im Small-OS verwendeten Konstanten (ohne Konstanten für Returncodes)

Konstante	Bedeutung
C_Error	Konstante für allgemeinen Fehler (z. B.: bei Aufruf eines Unterprogramms)
C_InvalidPointer	Konstante für einen ungültigen Zeigerwert eines Zeigers
C_Equal	Konstante für Vergleiche ODER verglichene Objekte sind gleich
C_NotEqual	Konstante für Vergleiche ODER verglichene Objekte sind ungleich
C_Found	Konstante für Suchfunktionen ODER gesuchtes Objekt gefunden
C_NotFound	Konstante für Suchfunktionen ODER gesuchtes Objekt nicht gefunden
C_AccessDenied	Konstante für Zugriffsbedingungen, Zugriff verweigert
C_AccessAllowed	Konstante für Zugriffsbedingungen, Zugriff erlaubt
C_WriteError	Konstante für Fehler beim Schreiben von Daten (z.B.: beim Schreiben ins EEPROM)
C_EFTypeWorking	Konstante für ein EF vom Typ „working EF“
C_EFTypeInternal	Konstante für ein EF vom Typ „internal EF“
C_EFStrucLinFix	Konstante für ein EF mit der Struktur „linear fixed“
C_EFStrucTransp	Konstante für ein EF mit der Struktur „transparent“
C_CmdVERIFY	Konstante für das Kommando VERIFY
C_CmdINTAUTH	Konstante für das Kommando INTERNAL AUTHENTICATE

Tabelle 5.36 Tabelle der im Small-OS verwendeten Konstanten für Returncodes

Konstante	Inhalt	Bedeutung des Returncodes
Process completed – Warning Processing		
C_RC_CounterX	'63Cx'	Zähler (in der Regel der Fehlbedienungs-zähler der PIN) wurde erhöht und hat nun den Stand „x“
Process aborted – Execution Error		
C_RC_MemoryFailure	'6581'	EEPROM-Schreibfehler
Process aborted – Checking Error		
C_RC_WrongLength	'6700'	falsche Länge
C_RC_CmdIncompFStruc	'6981'	Kommando ist inkompatibel zur Dateistruktur
C_RC_SecStateNotSatisfied	'6982'	Sicherheitszustand nicht erfüllt
C_RC_AuthMethodBlocked	'6983'	Authentisierungsmethode gesperrt
C_RC_CondOfUseNotSatisfied	'6985'	Die Bedingungen für die Benutzung (des Datenelements d. h. PIN oder Schlüssel) sind nicht erfüllt
C_RC_CmdNotAllowed	'6986'	Kommando nicht erlaubt
C_RC_FktNotSupported	'6A81'	Funktion wird nicht unterstützt
C_RC_FileNotFound	'6A82'	Datei nicht gefunden
C_RC_RecordNotFound	'6A83'	Record nicht gefunden
C_RC_LcInconsitentPIP2	'6A87'	Lc inkonsistent mit P1 oder P2
C_RC_RefDataNotFound	'6A88'	im Kommando referenzierte Daten nicht gefunden
C_RC_WrongPIP2	'6B00'	P1 oder P2 falsch
C_RC_INSNotSupported	'6D00'	Kommando wird nicht unterstützt
C_RC_CLANotSupported	'6E00'	Class wird nicht unterstützt
C_RC_FatalError	'6F00'	Konstante für Returncode: interner Fehler ohne nähere Bezeichnung
Process completed – Normal processing		
C_RC_OK	'9000'	Kommando erfolgreich ausgeführt

Tabelle 5.37 Tabelle der im Small-OS verwendeten Variablen für die Datenübertragung von und zur Chipkarte. Die aufgeführten Variablen werden typischerweise im RAM gehalten.

Variable	Beschreibung
.1 8	bitweises Lesen oder Schreiben einer Variablen (z. B.: APDU.Cmd.CLA.1 entspricht dem Bit 1 des Class Bytes in der Kommando-APDU)
APDU.Cmd	von der Chipkarte empfangene Kommando-APDU
APDU.Cmd.CLA	Class Byte der Kommando-APDU
APDU.Cmd.INS	Instruction Byte der Kommando-APDU
APDU.Cmd.P1	Parameter 1 der Kommando-APDU
APDU.Cmd.P2	Parameter 2 der Kommando-APDU
APDU.Cmd.Lc	Länge Kommando Byte der Kommando-APDU (optional)
APDU.Cmd.Data [. . .]	Datenteil der Kommando-APDU mit der Länge 1 . . . n (optional)
APDU.Cmd.Le	erwartete Länge der Antwort-APDU (optional)
APDU.Rsp	von der Chipkarte zu sendende Antwort-APDU
APDU.Rsp.Data [. . .]	Datenteil der Antwort-APDU mit der Länge 1 . . . n (optional)
APDU.Rsp.SW1	Statuswort 1 der Antwort-APDU (Byte 1 des Returncodes)
APDU.Rsp.SW2	Statuswort 2 der Antwort-APDU (Byte 2 des Returncodes)
Returncode	Returncode := APDU.Rsp.SW1 APDU.Rsp.SW2

Tabelle 5.38 Tabelle der im Small-OS verwendeten Variablen für Dateiverwaltung und Dateizugriff. Die aufgeführten Variablen werden typischerweise im RAM gehalten.

Variable	Beschreibung
Ptr.MF	Zeiger im Betriebssystem der Chipkarte, zeigt immer auf das MF
Ptr.CurrentDF	Zeiger im Betriebssystem der Chipkarte, zeigt auf das aktuell selektierte DF bzw. auf das MF
Ptr.CurrentIEF.Key	Zeiger im Betriebssystem der Chipkarte, zeigt auf das aktuelle internal EF mit den Schlüsseln im jeweils selektierten DF (EF Key)
Ptr.CurrentWEF	Zeiger im Betriebssystem der Chipkarte, zeigt auf das aktuell selektierte working EF im jeweils selektierten DF bzw. im MF
Ptr.CurrentRecord	Zeiger im Betriebssystem der Chipkarte, zeigt auf den aktuellen Record bei einem Record-orientierten EF. Der Zeiger ist bei einer Selektion einer transparenten Datei ungültig.
SecurityState.MF	erreichter Sicherheitszustand im MF
SecurityState.DF	erreichter Sicherheitszustand im aktuell selektierten DF

Tabelle 5.39 Tabelle der im Small-OS verwendeten generellen Datenstrukturen für die Dateiverwaltung. Die aufgeführten Variablen werden typischerweise im EEPROM gehalten (Teil 1).

Variablenname	Beschreibung, Inhalt und Größe
Datenstrukturen für das MF und die DFs	
DF[. . .].FID	File Identifier Länge: 2 Byte; Inhalt: jeweils 0 . . . 255
DF[. . .].DFName	DF Name (schließt den Application Identifier AID ein) Länge: 1 . . . 16 Byte; Inhalt: jeweils 0 . . . 255
DF[. . .].LenDFName	Länge des DF Name Länge: 1 Byte; Inhalt: Ganzzahl $\in \{1 \dots 16\}$

Tabelle 5.40 Tabelle der im Small-OS verwendeten generellen Datenstrukturen für die Dateiverwaltung. Die aufgeführten Variablen werden typischerweise im EEPROM gehalten (Teil 2).

Variablenname	Beschreibung, Inhalt und Größe
Datenstrukturen für EFs	
DF[. . .].EF[. . .]. . .	Dateibaum in Form einer 2-dimensionalen Matrix
DF[. . .].EF[. . .].FID	File Identifier Länge: 2 Byte; Inhalt: jeweils 0 . . . 255
DF[. . .].EF[. . .].Structure	Struktur des EFs Inhalt: Eigenschaft transparent / linear fixed
DF[. . .].EF[. . .].Type	Typ des EFs Inhalt: Eigenschaft working / internal
DF[. . .].EF[. . .].AccessCondition.Read	Zustand der Zugriffsbedingung, damit das Kommando READ auf das EF erlaubt ist. Inhalt: 0 . . . 255
DF[. . .].EF[. . .].AccessCondition.Update	Zustand der Zugriffsbedingung, damit das Kommando UPDATE auf das EF erlaubt ist. Inhalt: 0 . . . 255

Tabelle 5.41 Tabelle der im Small-OS verwendeten Datenstrukturen für die internal EF-Verwaltung zur Ablage von PINs und Schlüsseln in einem EF der Struktur „linear variable“. Die aufgeführten Variablen werden typischerweise im EEPROM gehalten.

Variablenname	Beschreibung, Inhalt und Größe
Datenstrukturen für EFs zur Ablage von Schlüsseln	
DF[...].EF[...].Record[x].KeyData	PIN/Schlüssel in einer Datei mit der Struktur „linear variable“ Länge: n (= KeySize); Inhalt: jeweils 0 ... 255
DF[...].EF[...].Record[x].KeySize	Länge der PIN / des Schlüssels in Byte
DF[...].EF[...].Record[x].KeyNo	Nummer der PIN / des Schlüssels Inhalt bei einer PIN: 1; 2 Inhalt bei einem Schlüssel: jeweils 1 ... 31
DF[...].EF[...].Record[x].RCntr	Fehlbedienungszähler zu einer PIN / einem Schlüssel Inhalt: 0 ... 15
DF[...].EF[...].Record[x].RCntrMaxValue	Maximalwert des Fehlbedienungszählers zu einer PIN/ einem Schlüssel Inhalt: 1 ... 15
DF[...].EF[...].Record[x].KeyPurpose	Verwendung des Dateninhalts des EF Key Records (VERIFY oder INTERNAL AUTHENTICATE)
DF[...].EF[...].Record[x].EntryState	Notwendiger Zustand zur Benutzung einer PIN/ eines Schlüssels Inhalt: 0 ... 255
DF[...].EF[...].Record[x].ResultState.OK	Zustand nach erfolgreicher Benutzung einer PIN/ eines Schlüssels Inhalt: 0 ... 255
DF[...].EF[...].Record[x].ResultState.NOK	Zustand nach fehlgeschlagener Benutzung einer PIN/ eines Schlüssels Inhalt: 0 ... 255

Tabelle 5.42 Tabelle der im Small-OS verwendeten Datenstrukturen für die Verwaltung der working EF mit den Strukturen „transparent“ und „linear fixed“. Die aufgeführten Variablen werden typischerweise im EEPROM gehalten.

Variablenname	Beschreibung, Inhalt und Größe
Datenstrukturen für EFs der Dateistruktur transparent	
DF[...].EF[...].TransparentData[1 ... n]	Dateninhalt einer Datei mit der Struktur „transparent“ Länge: n (= TransparentDataSize); Inhalt: jeweils 0 ... 255
DF[...].EF[...].TransparentDataSize	Größe des Dateninhalts einer Datei der Struktur „transparent“ Inhalt: 0 ... 255
Datenstrukturen für EFs der Dateistruktur linear fixed	
DF[...].EF[...].Record[...].Data[1 ... n]	Dateninhalt eines einzelnen Records einer Datei mit der Struktur „linear fixed“ Länge: 1 ... n (n = ...Size); Inhalt: jeweils 0 ... 255
DF[...].EF[...].Record[...].Size	Länge des einzelnen Records in einer Datei mit der Struktur „linear fixed“ (für alle Records gleich bei linear fixed) Inhalt: 1 ... 255
DF[...].EF[...].NoOfRecords	Anzahl der Records in einer Datei mit der Struktur „linear fixed“ Inhalt: 1 ... 255

Hauptschleife und Initialisierung von Hardware und Betriebssystem

Nach einem Reset der Chipkarte wird der Programmzähler mit der Adresse des Resetvektors geladen, und der Prozessor startet mit der Ausführung des ersten Programmcodes. Sofort wird, analog eines PCs, die Hardware auf Funktionstüchtigkeit geprüft. Dies umfasst zuerst die Prüfung des RAMs und anschließend die Berechnung von mehreren Prüfsummen über ROM und EEPROM. Tritt hier ein Fehler auf, dann versucht die Chipkarte, einen speziellen ATR (Fehler-ATR) zu senden, und nachfolgend wartet das Betriebssystem ohne weitere Aktionen in einer Endlosschleife auf den nächsten Reset. Ob das Senden dieses speziellen ATRs gelingt, hängt vor allem von dem aufgetretenen Fehler ab. Ist er schwerwiegender Natur und betrifft beispielsweise das RAM oder den Programmcode der Senderoutine, kann der ATR entweder nur noch verstümmelt oder überhaupt nicht mehr gesendet werden.

Nachdem die Chiphardware geprüft und initialisiert wurde, startet die Initialisierung des Betriebssystems. Die wesentlichen Punkte dabei sind das Einstellen der Übertragungsparameter für das T = 1 Protokoll, die automatische Selektion des MF und das Setzen der Sicherheitszustände für den Dateizugriff. Kann das MF nicht gefunden werden, ist dies ein schwerwiegender Fehler, und Small-OS stellt nach dem Senden eines Fehler-ATRs alle weiteren Aktivitäten ein.

I/O-Manager

Die gesamte serielle Datenübertragung vom und zum Terminal wird über den I/O-Manager abgewickelt. Die Grundlage zum Senden und Empfangen der Nachrichten sind zwei hardwareabhängige ProgrammROUTINEN im Kern des Betriebssystems („Kernel_IO_SendByte“, „Kernel_IO_ReceiveByte“). Alle anderen Teile des I/O-Programmcodes sind unabhängig von der Zielhardware des Small-OS. Der ATR und das Übertragungsprotokoll T = 1 werden sowohl für den Gutfall als auch für den Schlechtfall vollständig im I/O-Manager durchgeführt. Vor allem für die relativ komplizierten Abläufe des T = 1 Protokolls muss deshalb ein eigener Zustandsautomat realisiert sein. In der ISO/IEC 7816-3 Amd. 1 sind anhand vieler Beispiele die Reaktionen des I/O-Managers für alle Gut- und Schlechtfälle dargestellt. In handoptimiertem Assembler programmiert, benötigt ein guter I/O-Manager mindestens ein Kilobyte Speicher, üblicherweise im ROM.

Echte I/O-Manager benutzen vorzugsweise ausschließlich das RAM als I/O-Puffer, da nur hier Schreibzugriffe mit voller CPU-Geschwindigkeit möglich sind. Allerdings gibt es mittlerweile viele Chipkarten-Betriebssysteme, deren Empfangs- und Sendepuffer größer ist als das zur Verfügung stehende RAM. In diesem Fall wird ab einem bestimmten Schwellwert automatisch ein bestimmter Teil des EEPROMs als erweiterter Puffer für die Datenübertragung benutzt. Damit umgeht man alle Einschränkungen der Länge des I/O-Buffers, erkaufte sich dies aber aufgrund der EEPROM-Schreibzeiten mit einer erheblich reduzierten effektiven Datenübertragungsgeschwindigkeit. Zudem stellt sich bei dieser Art des erweiterten I/O-Puffers die Gefahr ein, dass aufgrund der häufigen Schreibzugriffe der EEPROM-Teil des Puffers in kurzer Zeit seinen Geist aufgibt. Trotz dieser Einschränkungen ist es der einzige technisch machbare Weg, um den Datenübertragungspuffer größer als ein zur Verfügung stehendes RAM zu machen.

File Manager

Bei echten Chipkarten-Betriebssystemen finden aus Sicherheitsgründen alle Dateizugriffe über einen zentralen File-Manager statt. Dieser hat oftmals auch noch die Funktion, Prüfsum-

men über den angesprochenen Dateihdr zur Zugriffszeit zu berechnen. Als Zugestndnis an die Einfachheit wird bei der Kommandoabarbeitung von Small-OS direkt ohne Schichten-trennung und vorherige Prfsummenberechnung auf die mehrdimensionalen Variablen des Dateibaums zugegriffen. Lediglich die Prfung auf erlaubten Dateizugriff wird fr alle Kommandos ber den File-Manager durchgefhr. Den Lese- und Schreibzugriff auf Dateien mit einem vollstndigen File-Manager kann man sich hnlich vorstellen.

Betriebssystemkern

Alle Programmteile, die zum Kern des Betriebssystems gehren, sind unter „OS-Kernel“ zusammengefasst. Ein Grobteil dieser Routinen ist entweder hardwareabhngig oder betrchtlich zeitkritisch und muss deshalb bei einer Portierung auf eine andere Hardwareplattform adaptiert werden. Die Funktionalitt dieser Routinen ist jeweils durch den Namen beschrieben. Manche dieser Unterprogramme haben Einflsse auf sicherheits-technische Belange. So drfen beispielsweise in einem Unterprogramm wie „Kernel_CompareByte-String“ keine Laufzeitunterschiede auftreten, die abhngig vom Vergleichsergebnis sind, da dies ein mglicher Ansatzpunkt wre, um mittels Zeitmessungen interne Ergebnisse einer PIN-Berechnung zu ermitteln. Mittlerweile wird diese Art von Angriff zwar dadurch eliminiert, dass der Fehlbedienungszhler prophylaktisch vor jedem PIN-Vergleich erhht wird, doch noch vor einigen Jahren wre dies ein sehr aussichtsreicher Angriff auf die PIN gewesen.

Kommandointerpreter

Der Kommandointerpreter bei Small-OS ist verhltnismssig einfach aufgebaut. Er stellt dem Prinzip nach einen Verteiler dar, der abhngig von Class und Instruction einen Programmcode aufruft, in dem dann das erkannte Kommando abgearbeitet wird. Diese Realisierung hat neben dem geringen Speicherverbrauch einen weiteren wichtigen Vorteil. Es ist damit verhltnismssig einfach, neue Kommandos in das Betriebssystem zu integrieren. Dazu mssen im Kommandointerpreter wenige Zeilen Programmcode mit der neuen Instruction mit einem entsprechenden Unterprogrammaufruf eingefgt werden. Anschlieend wird das neue Kommando erkannt und je nach Bedarf ausgefhrt.

Die in der Praxis gebruchlichen Kommandointerpreter sind jedoch weitaus komplizierter aufgebaut, da sie sich einerseits im unvernderbaren ROM befinden, aber andererseits die Mglichkeit vorhanden sein muss, bei der Komplettierung des Betriebssystems noch Programmcode im EEPROM nachzuladen. Dieser nachtrglich geladene Programmcode muss dann zur Laufzeit erkannt und aufgerufen werden knnen. Das dabei zur Anwendung kommende Prinzip ist eine im EEPROM befindliche Sprungtabelle, die dann bei der Komplettierung je nach Bedarf ergnzt wird.

Eine feste Abfrage auf ein bestimmtes Class Byte wie in diesem Kommandointerpreter ist nur bei Betriebssystemen sinnvoll, die weder Secure Messaging noch logische Kanle untersttzen und nur eine einzige ISO/IEC7816-4 konforme Kommandoklasse untersttzen. In allen anderen Fllen wird das Class Byte nmlich zur Kennzeichnung vorgenannter Optionen benutzt und muss nicht mehr zwangslufig fr alle Kommandos einheitlich sein.

Returncode Manager

Die Aufgabe des Returncode Managers ist es, anhand eines bergebenen Werts fr den Returncode diesen aus einer Tabelle zu lesen und dann am Ende eines eventuell vorhandenen

Datenteils im Sendepuffer anzulegen. Dafür verwendet man vor allem aus zwei Gründen einen eigenen Manager: Zuerst ist es programmiertechnisch günstiger, wenn alle Returncodes an einer Stelle zentral gesetzt werden. Damit hätte man im Falle eines Softwarefehlers beispielsweise die Möglichkeit, einen Returncode auf einfache Weise und an einem zentralen Punkt durch einen anderen Returncode zu substituieren. Dies kann manchmal durchaus erforderlich sein. Der zweite Grund zum Einsatz eines Returncode Managers ist wieder einmal der rare Speicherplatz. Es erfordert berechenbar weniger Speicher, wenn die Returncodes gesammelt jeweils nur einmal in einer Tabelle abgelegt sind, als sie an jeder benötigten Stelle im Programmcode wiederholt aufzuführen.

Tabelle 5.43 Programmcode des Small-OS: Start des Betriebssystems und Hauptschleife

Resetvektor: CALL Initialize_Hardware CALL Initialize_Operating_System CALL IO_Manager_Send_ATR	Einsprungstelle nach einem Reset der CPU
Main_Loop: CALL IO_Manager_Receive_APDU CALL Command_Interpreter CALL IO_Manager_Send_APDU Goto Main_Loop	Hauptschleife des Chipkarten-Betriebssystems

Tabelle 5.44 Programmcode des Small-OS: Routinen für die Datenübertragung zur und von der Chipkarte

IO_Manager IO_Manager_Send_Error_ATR: Sende einen ATR, der einen schwerwiegenden Betriebssystemfehler anzeigt Error_ATR: GOTO Error_ATR	Hardware-nahe Unterprogramme für den Datenaustausch über die serielle I/O-Leitung mit dem Terminal Unterprogramm zum Aussenden eines speziellen ATRs, der schwerwiegende Fehler der Chipkartenhardware oder des Betriebssystems anzeigt. Anschließend wird das Betriebssystem durch eine Endlosschleife angehalten.
IO_Manager_Send_ATR: ... RETURN	Unterprogramm, das alle Parameter des Übertragungsprotokolls in einem ATR-String codiert und diesen aussendet. Verwendet die „Kernel_IO_SendByte“-Routine des Betriebssystemkerns.
IO_Manager_Send_APDU: ... RETURN	Unterprogramm, das eine im I/O-Puffer bereitgestellte APDU entsprechend den gewählten Übertragungsparametern in eine TPDU umwandelt und dann über die serielle I/O-Leitung zum Terminal sendet. Bei der Datenübertragung werden wenn nötig die Mechanismen zur Korrektur von Übertragungsfehlern angewandt. Verwendet die „Kernel_IO_SendByte“-Routine des Betriebssystemkerns.

Tabelle 5.44 (Fortsetzung)

IO_Manager_Receive_APDU: ... APDU.Cmd := vom Terminal empfangene und _ konvertierte TPDU ... RETURN	Unterprogramm, das über die serielle I/O-Leitung entsprechend den gewählten Übertragungsparametern eine TPDU vom Terminal empfängt, in eine APDU umwandelt und dann im I/O-Puffer ablegt. Bei der Datenübertragung werden wenn nötig die Mechanismen zur Korrektur von Übertragungsfehlern angewandt. Verwendet die „Kernel IO Receive Byte“ Routine des Betriebssystemkerns.
IO_Manager_Set_Transmission_Parameter: ... setze Clock Rate Conversion Factor auf 372 setze Convention setze Länge des Sendepuffers setze Länge des Empfangspuffers ... RETURN	Unterprogramm, das die Parameter für die serielle Datenübertragung setzt.

Tabelle 5.45 Programmcode des Small-OS: Initialisierung der Mikrocontroller-Hardware und des Chipkarten-Betriebssystems

Initialize_Hardware: CALL Kernel_CheckRAM IF STATUS(Kernel_CheckRAM = C_Error) THEN (_ GOTO IO_Manager_Send_Error_ATR) CALL Kernel_DeleteRAM IF STATUS(Kernel_DeleteRAM = C_Error) THEN (_ GOTO IO_Manager_Send_Error_ATR) CALL Kernel_Check_EDC_ROM IF STATUS(Kernel_Check_EDC_ROM = C_Error) THEN (_ GOTO IO_Manager_Send_Error_ATR) CALL Kernel_Check_EDC_EEPROM IF STATUS(Kernel_Check_EDC_EEPROM = C_Error) _ THEN (_GOTO IO_Manager_Send_Error_ATR) RETURN	Unterprogramm für die Initialisierung der Chipkarten-Hardware Prüfe das RAM auf Funktionsfähigkeit. Setze das gesamte RAM auf den Wert '00', um nach jedem Reset einen definierten Anfangszustand zu erreichen. Als beabsichtigter Nebeneffekt werden damit alle Variablen im RAM initialisiert. Prüfe an verschiedenen Stellen Fehlererkennungs-codes (EDC) im ROM. Prüfe an verschiedenen Stellen Fehlererkennungs-codes (EDC) im EEPROM.
Initialize_Operating_System: CALL IO_Manager_Set_Transmission_Parameter CALL SELECT_FILE_MF IF STATUS(SELECT_FILE_MF = C_Error) THEN (_ GOTO IO_Manager_Send_Error_ATR) SecurityState.MF := 0 SecurityState.DF := 0 RETURN	Unterprogramm für die Initialisierung des Chipkarten-Betriebssystems

Tabelle 5.46 Programmcode des Small-OS: Hardware-abhängige Routinen des Betriebssystemkerns.

OS-Kernel	Hardware-nahe Unterprogramme des Betriebssystemkerns
Kernel_Check_EDC_x: ... RETURN	Unterprogramme des Betriebssystemkerns zur Berechnung von Prüfsummen (EDCs), um eine Konsistenzprüfung von Speicherbereichen im ROM oder EEPROM durchzuführen. x · {ROM, EEPROM}
Kernel_x: ... RETURN	Unterprogramme des Betriebssystemkerns, für elementare Funktionen. x · {CopyByteString, CompareByteString, DeleteRAM, CheckRAM}
Kernel_DES_x: ... RETURN	Unterprogramme des Betriebssystemkerns, für Ver- und Entschlüsselung mittels DES. x · {Encrypt, Decrypt}
Kernel_IO_x: ... RETURN	Unterprogramme des Betriebssystemkerns, für das Senden und Empfangen eines einzelnen Bytes über die serielle Schnittstelle des Mikrocontrollers. x · {SendByte, ReceiveByte}

Tabelle 5.47 Programmcode des Small-OS: Kommandointerpreter.

Command Interpreter:	Kommandointerpreter
<pre>IF APDU.Cmd.CLA <> '00' THEN (Returncode := C_RC_CLANotSupported GOTO Command_Interpreter_Exit) IF APDU.Cmd.INS = 'A4' THEN (CALL SELECT_FILE GOTO Command_Interpreter_Exit) IF APDU.Cmd.INS = 'B0' THEN (CALL READ_BINARY GOTO Command_Interpreter_Exit) IF APDU.Cmd.INS = 'D6' THEN (CALL UPDATE_BINARY GOTO Command_Interpreter_Exit) IF APDU.Cmd.INS = 'B2' THEN (CALL READ_RECORD GOTO Command_Interpreter_Exit) IF APDU.Cmd.INS = 'DC' THEN (CALL UPDATE_RECORD GOTO Command_Interpreter_Exit) IF APDU.Cmd.INS = '20' THEN (CALL VERIFY GOTO Command_Interpreter_Exit) IF APDU.Cmd.INS = '88' THEN (CALL INTERNAL_AUTHENTICATE GOTO Command_Interpreter_Exit) Returncode := C_RC_INSNotSupported Command_Interpreter_Exit: CALL Returncode_Manager RETURN</pre>	Falls das Class Byte ungleich der Class für ISO/IEC 7816-4 Kommandos ist, brich die weitere Kommandobearbeitung ab und setze den entsprechenden Returncode. Falls das Kommando SELECT FILE zur Chipkarte gesendet wurde, rufe das entsprechende Unterprogramm auf. Falls das Kommando READ BINARY zur Chipkarte gesendet wurde, rufe das entsprechende Unterprogramm auf. Falls das Kommando UPDATE BINARY zur Chipkarte gesendet wurde, rufe das entsprechende Unterprogramm auf. Falls das Kommando READ RECORD zur Chipkarte gesendet wurde, rufe das entsprechende Unterprogramm auf. Falls das Kommando UPDATE RECORD zur Chipkarte gesendet wurde, rufe das entsprechende Unterprogramm auf. Falls das Kommando VERIFY zur Chipkarte gesendet wurde, rufe das entsprechende Unterprogramm auf. Falls das Kommando INTERNAL AUTHENTICATE zur Chipkarte gesendet wurde, rufe das entsprechende Unterprogramm auf. Das an die Chipkarte übergebene Kommando wird nicht unterstützt. Setze im I/O-Puffer den vom Programmcode des Kommandos festgelegten Returncode.

Tabelle 5.48 Programmcode des Small-OS: Returncode Manager und File Manager.

Returncode_Manager: ... RETURN	Manager zum Setzen der Returncodes Setzt Returncode anhand des übergebenen Wertes in Returncode und einer Tabelle.
File_Manager_CheckACRead: Status := C_AccessDenied IF Ptr.CurrentDF = Ptr.MF THEN (// MF ist selektiert IF .EF[Ptr.CurrentWEF].AccessCondition.Read = _ SecurityState.MF THEN (Status := C_AccessAllowed) ELSE (// DF ist selektiert IF .EF[Ptr.CurrentWEF].AccessCondition.Read = _ SecurityState.DF THEN (Status := C_AccessAllowed)) RETURN	Dateimanager für die Prüfung der Dateizugriffsbedingung „Read“ Setze die Variable für den Zugriffsstatus auf den Ausgangswert. Prüfe, ob der Sicherheitszustand erreicht ist, damit aus dem selektierten EF mit READ BINARY oder READ RECORD gelesen werden darf. Es muss abhängig vom aktuell selektierten Verzeichnis (d. h. MF oder DF) der Sicherheitszustand erreicht sein.
File_Manager_CheckACUpdate: Status := C_AccessDenied IF Ptr.CurrentDF = Ptr.MF THEN (// MF ist selektiert IF .EF[Ptr.CurrentWEF].AccessCondition.Update = _ SecurityState.MF THEN (Status := C_AccessAllowed) ELSE (// DF ist selektiert IF .EF[Ptr.CurrentWEF].AccessCondition.Update = _ SecurityState.DF THEN (Status := C_AccessAllowed)) RETURN	Dateimanager für die Prüfung der Dateizugriffsbedingung „Update“ Setze die Variable für den Zugriffsstatus auf den Ausgangswert. Prüfe, ob der Sicherheitszustand erreicht ist, damit aus dem selektierten EF mit UPDATE BINARY oder UPDATE RECORD gelesen werden darf. Es muss abhängig vom aktuell selektierten Verzeichnis (d. h. MF oder DF) der Sicherheitszustand erreicht sein.

Grundsätzlicher Aufbau des Programmcodes eines Chipkarten-Kommandos

Eine Beschreibung der grundlegenden Funktionalität von Chipkarten-Kommandos befindet sich in Kapitel 7, „Kommandos von Chipkarten“. Der Programmcode für die Kommandoabarbeitung ist in seiner Struktur immer dreigeteilt und untereinander durch dünne Linien abgetrennt. Zu dem Kommando gehörende Unterprogramme sind durch dicke Linien vom eigentlichen Programmcode separiert. Im ersten Funktionsblock wird der Kommando-Header (d. h. CLA, INS, P1, P2) so weit als zu diesem Zeitpunkt möglich untersucht. Dabei wird dem in der Informatik weithin bekannten Paradigma Rechnung getragen, dass Daten so früh wie möglich auf Konsistenz und Bereiche abzu prüfen sind.

Anschließend findet eine Kontrolle der grundlegenden Voraussetzungen zur Ausführung des Kommandos statt. Hier findet dann beispielsweise die Überwachung der Dateistruktur oder der Zugriffsbedingungen auf die Datei statt. Sind alle diese Prüfungen ohne Zurückweisung durchlaufen, findet die tatsächliche Kommandoausführung statt. Vom Codeumfang her ist diese meist sehr kurz. Am Schluss wird dann der Returncode für ein erfolg-

reich ausgeführtes Kommando gesetzt und über Kommandointerpreter und Returncode-Manager zum I/O-Manager gesprungen. Dieser sendet das Ergebnis der Kommandoabarbeitung und wartet anschließend auf ein neues Kommando.

Tritt bei irgendeiner Prüfung im Pseudocode ein Fehler auf, wird sogleich der dazugehörige Returncode gesetzt und auf den zentralen Ausgang des betreffenden Unterprogramms verzweigt.

Befehlssatz

Alle sieben im Folgenden aufgeführten Kommandos entsprechen in Codierung und Aufbau der ISO/IEC 7816-4. Aufgrund der Freiheiten in der Implementierung, die eine Norm naturgemäß lässt, ist es jedoch notwendig, jedes Kommando zusätzlich zur Referenz auf die Norm im Detail noch einmal zu spezifizieren. Die folgenden Zeilen sind deshalb die Spezifikation aller Kommandos von Small-OS. Sie sind relativ kurz und formal gehalten und stellen deshalb nur das Wesentliche dar.

Kommando SELECT FILE

Das Kommando SELECT FILE wird benutzt, um eine Datei (MF, DF, EF) auszuwählen. Dazu ist im Regelfall der 2 Byte lange FID (*file identifier*) notwendig. Im Profil N der ISO/IEC 7816-4 ist darüber hinaus vorgesehen, dass DFs auch mit dem DF Namen, der einen AID (*application identifier*) enthalten kann, ausgewählt werden können. Der AID darf bei Small-OS nur als Ganzes übergeben werden. Im Sonderfall des MFs benötigt man zur Auswahl keine FID, sondern nur die entsprechende Option im Kommando. Das MF ist nach einem Reset der Chipkarte automatisch selektiert und kann innerhalb der Sitzung auch von jedem anderen Verzeichnis aus selektiert werden.

SELECT FILE fällt bei direkter Selektion des MFs unter Case 1, was bedeutet, dass weder bei der Kommando-APDU noch bei der Antwort-APDU ein Datenteil vorhanden ist. Bei der Selektion mittels FID oder DF Name fällt SELECT FILE unter Case 3, d. h. es ist in der Kommando-APDU ein Datenteil vorhanden, jedoch nicht in der Antwort-APDU.

Der Sicherheitszustand nach dem Reset ist im MF auf den Grundzustand „0“ gesetzt. Die Selektion eines DF hat keine Auswirkungen auf den Sicherheitszustand im MF, im selektierten DF wird er aber automatisch auf den Grundzustand „0“ gesetzt. Bei der Selektion eines EFs der Struktur „linear fixed“ wird der Record-Zeiger in den Zustand „ungültig“ gesetzt.

Die Reihenfolge, ob bei der Selektion mit FID zuerst ein DF oder ein EF gesucht wird, ist nicht durch eine Norm festgelegt. Allerdings ist sie in bestimmten Fällen von erheblicher Bedeutung. Ist beispielsweise das MF selektiert, und es existiert sowohl ein DF als auch ein EF mit gleicher FID, hängt es nur von der Suchroutine ab, ob das EF unter Umständen nie selektiert werden kann. Deshalb wurde bei der Implementierung von Small-OS die Reihenfolge in der Suchroutine so gewählt, dass bei der Selektion mit FID immer zunächst in der Liste der EFs gesucht wird und erst dann in der Liste der DFs. Ein DF kann im Konfliktfall immer via DF Name selektiert werden, sodass die Benutzung des FIDs hier nicht zwangsläufig notwendig ist.

Tabelle 5.49 Small-OS: Codierung des Case-1-Kommandos SELECT FILE bei der Option „Direkte Selektion des MF“

Datenelement	Codierung	Bemerkung
CLA	'00'	---
INS	'A4'	---
P1	'00'	---
P2	'00'	---

Tabelle 5.50 Small-OS: Codierung des Case-3-Kommandos SELECT FILE bei der Option „Selektion einer Datei mit dem FID“

Datenelement	Codierung	Bemerkung
CLA	'00'	---
INS	'A4'	---
P1	'00'	---
P2	'00'	---
Lc	2	---
DATA	FID	Angabe des 2 Byte langen FID für MF, DF oder EF

Tabelle 5.51 Small-OS: Codierung des Case-3-Kommandos SELECT FILE bei der Option „Selektion eines DFs mit DF Name“

Datenelement	Codierung	Bemerkung
CLA	'00'	---
INS	'A4'	---
P1	'04'	---
P2	'00'	---
Lc	1 ... 16	---
DATA	DF Name	Angabe des 1 bis 16 Byte langen DF Name, der den AID der zu selektierenden Anwendung enthalten kann.

Tabelle 5.52 Small-OS: Codierung der Antwort auf das Kommando SELECT FILE

Datenelement	Codierung	Bemerkung
SW1 SW2	'9000'	Returncode für den Fall des korrekt ausgeführten Kommandos.

Tabelle 5.53 Small-OS: Programmcode des Small-OS: Kommando SELECT FILE nach ISO 7816-4 Profil

SELECT_FILE:	Kommando nach ISO/IEC 7816-4, Profil N mit Erweiterungen
<pre>IF APDU.Cmd.P1 = '00' THEN (IF ((APDU.Cmd.P2 = '00') AND (Length(APDU.Cmd) = 4)) _ THEN (IF LENGTH (APDU.Cmd) < 4 THEN (Returncode := C_RC_WrongLength RETURN) CALL SELECT_FILE_MF RETURN) IF APDU.Cmd.Lc = '02' THEN (IF LENGTH (APDU.Cmd) < 6 THEN (Returncode := C_RC_WrongLength RETURN) IF APDU.Cmd.Data[1 ... 2] = '3F00' THEN (CALL SELECT_FILE_MF) RETURN) ELSE (CALL SELECT_FILE_FID) RETURN)) IF APDU.Cmd.P1 = °0000 0100° THEN (IF LENGTH (APDU.Cmd) < 5 THEN (Returncode := C_RC_WrongLength RETURN) IF ((APDU.Cmd.Lc < 1) OR (APDU.Cmd.Lc > 16)) THEN (Returncode := C_RC_LcInconsitentP1P2 RETURN) CALL SELECT_FILE_DFName) RETURN</pre>	Falls P1 = '00', handelt es sich entweder um die Kommandooption der direkten Selektion des MFs oder um die Selektion einer Datei mittels des 2 Byte langen FID. Falls P2 = '00' und nur der 4 Byte lange Kommando-Header übergeben wurde, handelt es sich um die direkte Selektion des MFs. Prüfe, ob das Kommando als Case-1-Kommando gesendet wurde. Ist dies nicht der Fall, setze den entsprechenden Returncode und brich die Kommandobearbeitung ab. Falls 2 Byte Daten übergeben wurden, soll eine Datei mit dem 2 Byte langem FID selektiert werden. Prüfe, ob das Kommando als Case-3-Kommando gesendet wurde. Ist dies nicht der Fall, setze den entsprechenden Returncode brich die Kommandobearbeitung ab. Bei einer Selektion des MFs (mit FID = '3F00') wird diese unmittelbar ausgeführt. Bei einer anderen Datei (DF oder EF) wird die Selektion über eine Suchroutine mit dem übergebenen FID als Suchwert ausgeführt. Prüfe, ob ein DF mit dem DF Name selektiert werden soll. Prüfe, ob das Kommando als Case-3-Kommando gesendet wurde. Ist dies nicht der Fall, setze den entsprechenden Returncode und brich die Kommandobearbeitung ab. Prüfe, ob der dazugehörige Datenteil zwischen 1 und 16 Bytes lang ist.

Tabelle 5.53 (Fortsetzung)

<pre> SELECT_FILE_MF: SEARCH (MF im Dateibaum) IF STATUS(SEARCH = C_NotFound) THEN (// MF im Dateibaum nicht gefunden Returncode := C_RC_FatalError RETURN) // MF im Dateibaum gefunden Ptr.CurrentDF := ermittelte MF-Adresse SEARCH (EF Key im MF) IF STATUS(SEARCH) = C_Found THEN (Ptr.CurrentIEF.Key := ermittelte EF Key-Adresse) ELSE (Ptr.CurrentIEF.Key := C_InvalidPointer) Ptr.CurrentWEF := C_InvalidPointer Ptr.CurrentRecord := C_InvalidPointer SecurityState.MF := 0 SecurityState.DF := 0 Returncode := C_RC_OK RETURN </pre>	Unterprogramm für die Selektion des MF Falls das MF im Dateibaum nicht gefunden werden konnte, brich mit internem Betriebssystemfehler ab. Setze den Zeiger des aktuellen DFs auf das gefundene MF. Suche ein EF Key, und setze beim Vorhandensein den entsprechenden Zeiger. Kennzeichne den Zeiger für das aktuelle EF und den Zeiger auf dem aktuellen Record als ungültig. Setze die Sicherheitszustände für MF und DF auf den Ausgangswert.
<pre> SELECT_FILE_FID: SEARCH (EF mit FID in dem aktuell selektiertem DF) IF STATUS(SEARCH) = C_Found THEN (Ptr.CurrentWEF := ermittelte EF Adresse Ptr.CurrentRecord := C_InvalidPointer Returncode := C_RC_OK RETURN) // EF mit der übergebenen FID nicht gefunden SEARCH (DF mit FID) IF STATUS(SEARCH) = C_Found THEN (Ptr.CurrentDF := ermittelte DF-Adresse SEARCH (EF Key im aktuellen DF) IF STATUS(SEARCH) = C_Found) THEN (Ptr.CurrentIEF.Key := ermittelte EF Key-Adresse) ELSE (Ptr.CurrentIEF.Key := C_InvalidPointer) Ptr.CurrentWEF := C_InvalidPointer Ptr.CurrentRecord := C_InvalidPointer SecurityState.DF := 0 Returncode := C_RC_OK RETURN)) // weder EF noch DF mit der übergebenen FID gefunden Returncode := C_RC_FileNotFound RETURN </pre>	Unterprogramm für die Selektion eines DF oder EF mit dem FID Suche im aktuellen DF nach einem EF mit der übergebenen FID. Falls ein EF gefunden werden konnte, setze den Zeiger des aktuellen EFs. Konnte das EF nicht gefunden werden, suche nach einem DF mit der übergebenen FID. Falls ein DF gefunden werden konnte, setze den Zeiger des aktuellen DFs darauf. Suche dann ein EF Key, und setze beim Vorhandensein den entsprechenden Zeiger. Kennzeichne den Zeiger für das aktuelle EF und den Zeiger auf dem aktuellen Record als ungültig. Setze den Sicherheitszustand für das DF auf den Ausgangswert. Konnte keine passende Datei gefunden werden, setze einen entsprechenden Returncode.
<pre> SELECT_FILE_DFName: SEARCH (DF mit DF Name) IF STATUS(SEARCH) = C_NotFound THEN (Returncode := C_RC_FileNotFound RETURN) ELSE (Ptr.CurrentDF := ermittelte DF-Adresse) SEARCH (EF Key im aktuellen DF) </pre>	Unterprogramm für die Selektion eines DF mit dem DF Name Setze den Zeiger auf das aktuelle DF auf das im Dateibaum gefundene DF. Suche dann das im aktuellen DF relevante EF Key und setze bei Vorhandensein den entsprechenden Zeiger darauf.

Tabelle 5.53 (Fortsetzung)

<pre>IF STATUS(SEARCH) = C_Found) THEN (Ptr.CurrentIEF.Key := ermittelte EF Key-Adresse) ELSE (Ptr.CurrentIEF.Key := C_InvalidPointer) Ptr.CurrentWEF := C_InvalidPointer Ptr.CurrentRecord := C_InvalidPointer SecurityState.DF := 0 Returncode := C_RC_OK RETURN</pre>	Kennzeichne den Zeiger für das aktuelle EF und den Zeiger auf dem aktuellen Record als ungültig. Setze den Sicherheitszustand für das DF auf den Ausgangswert.

Kommando READ BINARY

Mit dem Kommando READ BINARY können aus einem EF mit der Struktur „transparent“ Daten ab einem zu übergebenden 15 Bit großen Offset gelesen werden. READ BINARY fällt unter Case 2, d. h. bei der Kommando-APDU ist kein Datenteil vorhanden, jedoch bei der Antwort-APDU.

Alle Längenangaben müssen ein Vielfaches von einem Byte sein. Die maximale Länge der zu lesenden Daten ist auf die maximale Dateigröße eines transparenten EFs von 255 Byte beschränkt. Wird als Länge der Wert null übergeben, so werden ab dem übergebenen Offset alle Daten bis zum Ende der Datei gelesen. Im Profil N von ISO/IEC 7816-4 ist die implizite Selektion von EFs mittels Short-FID nicht vorgesehen und deshalb hier auch nicht realisiert.

Um Daten aus einem EF mit diesem Kommando zu lesen, muss die dazugehörige Zugriffsbedingung erfüllt sein, sonst wird das Kommando mit einer entsprechenden Fehlermeldung abgewiesen.

Tabelle 5.54 Small-OS: Codierung des Case-2-Kommandos READ BINARY

Datenelement	Codierung	Bemerkung
CLA	'00'	---
INS	'B0'	---
P1	°0XXX XXXX°	Angabe der höherwertigen 7 Bit des Offsets zu dem zu lesenden Datenteil. (Offset := XXX XXXX Y)
P2	Y	Angabe des niederwertigen Bytes des Offsets zu dem zu lesenden Datenteil.
Le	Z	Z = 0: Lese alle Bytes bis zum Ende der Datei. Z > 0: Z ist die Anzahl der zu lesenden Bytes.

Tabelle 5.55 Small-OS: Codierung der Antwort auf das Kommando READ BINARY

Datenelement	Codierung	Bemerkung
DATA	...	Falls das Kommando korrekt ausgeführt wurde, befinden sich in diesem Datenelement die mit dem Kommando angeforderten Daten.
SW1 SW2	'9000'	Returncode für den Fall des korrekt ausgeführten Kommandos.

Tabelle 5.56 Programmcode des Small-OS: Kommando READ BINARY nach ISO 7816-4 Profil N

<pre>READ_BINARY: IF LENGTH (APDU.Cmd) < 5 THEN (Returncode := C_RC_WrongLength RETURN) IF APDU.Cmd.P1.b8 = °1° THEN (Returncode := C_RC_FktNotSupported GOTO READ_BINARY_Exit) IF Ptr.CurrentWEF = C_InvalidPointer THEN (Returncode := C_RC_CmdNotAllowed GOTO READ_BINARY_Exit) WITH DF[Ptr.CurrentDF]. IF .EF[Ptr.CurrentWEF].Structure = C_EFStrucTransparent THEN (Returncode := C_RC_CmdIncompFStruc GOTO READ_BINARY_Exit) FileOffset := (APDU.Cmd.P1 * 256) + APDU.Cmd.P2 DataLenToRead := 0 IF Status(File_Manager_CheckACRead) = _ C_AccessDenied) THEN (Returncode := C_RC_SecStateNotSatisfied GOTO READ_BINARY_Exit) IF APDU.Cmd.Le = '00' THEN (DataLenToRead := .EF[Ptr.CurrentWEF]. _ TransparentDataSize - FileOffset) ELSE (DataLenToRead := APDU.Cmd.Le) IF .EF[Ptr.CurrentWEF].TransparentDataSize < FileOffset THEN (Returncode := C_RC_WrongP1P2 GOTO READ_BINARY_Exit) IF .EF[Ptr.CurrentWEF].TransparentDataSize < (FileOffset + DataLenToRead) THEN (Returncode := C_RC_WrongP1P2 GOTO READ_BINARY_Exit) CALL Kernel_CopyByteString // Von: .EF[Ptr.CurrentWEF].TransparentData[x ... y] // x = FileOffset ; y = (FileOffset + DataLenToRead) // Nach: APDU.Rsp.Data[1 ... DataLenToRead] Returncode := C_RC_OK READ_BINARY_Exit: END WITH RETURN</pre>	Kommando nach ISO/IEC 7816-4, Profil N Prüfe, ob das Kommando als Case-2-Kommando gesendet wurde. Ist dies nicht der Fall, setze den entsprechenden Returncode und brich die Kommandobearbeitung ab. Prüfe, ob ein EF mit Short-FID selektiert werden soll. Prüfe, ob bereits ein EF selektiert ist. Falls nein, dann Abbruch des Kommandos. Setze einen Teil des Dateibaums als Referenz für dieses Kommando. Prüfe, ob das selektierte EF die Dateistruktur „transparent“ hat. Berechne Offset zu den gewünschten Daten in der Datei und initialisiere die Variable für die Anzahl der zu lesenden Daten. Prüfe, ob der Sicherheitszustand erreicht ist, damit aus dem selektierten EF mit READ BINARY gelesen werden darf. Prüfe, ob alle verfügbaren Daten ausgelesen werden sollen (d. h. Le = '00') oder nur eine bestimmte Anzahl Daten. Prüfe, ob der angeforderte Offset zur Dateigröße passt. Prüfe, ob gewählter Offset und angeforderte Datenlänge (= Le) zur Dateigröße passen. Kopiere die angeforderten Daten von der Datei in den I/O-Sendepuffer. Das Kommando wurde fehlerfrei abgearbeitet, da in allen anderen Fällen der Fehleraussprung benutzt wurde.

Kommando UPDATE BINARY

Mit dem Kommando UPDATE BINARY können in ein EF mit der Struktur „transparent“ Daten ab einem zu übergebenden 15 Bit großen Offset geschrieben werden. UPDATE BINARY fällt unter Case 3, was bedeutet, dass bei der Kommando-APDU ein Datenteil vorhanden ist, jedoch nicht bei der Antwort-APDU.

Alle Längenangaben müssen ein Vielfaches von einem Byte sein. Die maximale Länge der zu schreibenden Daten ist auf die maximale Dateigröße eines transparenten EFs von 255 Byte beschränkt. Im Profil N von ISO/IEC 7816-4 ist die implizite Selektion von EFs mittels Short-FID nicht vorgesehen und deshalb hier auch nicht realisiert.

Um Daten in ein EF mit diesem Kommando zu schreiben, muss die dazugehörige Zugriffsbedingung erfüllt sein, sonst wird das Kommando mit einer entsprechenden Fehlermeldung abgewiesen.

Tabelle 5.57 Small-OS: Codierung des Case-3-Kommandos UPDATE BINARY

Datenelement	Codierung	Bemerkung
CLA	'00'	---
INS	'D6'	---
P1	°0XXX XXXX°	Angabe der höherwertigen 7 Bit des Offsets zu dem zu schreibenden Datenteil. (Offset := XXX XXXX Y)
P2	Y	Angabe des niederwertigen Bytes des Offsets zu dem zu schreibenden Datenteil.
Lc	...	Anzahl der zu schreibenden Bytes.
DATA	...	Die zu schreibenden Bytes.

Tabelle 5.58 Small-OS: Codierung der Antwort auf das Kommando UPDATE RECORD

Datenelement	Codierung	Bemerkung
SW1 SW2	'9000'	Returncode für den Fall des korrekt ausgeführten Kommandos.

Tabelle 5.59 Programmcode des Small-OS: Kommando UPDATE BINARY nach ISO 7816-4 Profil N

UPDATE_BINARY: IF LENGTH (APDU.Cmd) < 6 THEN (Returncode := C_RC_WrongLength RETURN) IF APDU.Cmd.P1.b8 = °1° THEN (Returncode := C_RC_FktNotSupported GOTO UPDATE_BINARY_Exit)	Kommando nach ISO/IEC 7816-4, Profil N Prüfe, ob das Kommando als Case-3-Kommando gesendet wurde. Ist dies nicht der Fall, setze den entsprechenden Returncode und brich die Kommandobearbeitung ab. Prüfe, ob ein EF mit Short-FID selektiert werden soll (d.h. das höherwertigste Bit (msb) von P1 ist gesetzt).

Tabelle 5.59 (Fortsetzung)

<pre>IF Ptr.CurrentWEF = C_InvalidPointer THEN (Returncode := C_RC_CmdNotAllowed GOTO UPDATE_BINARY_Exit) WITH DF[Ptr.CurrentDF]. IF .EF[Ptr.CurrentWEF].Structure <> C_EFStrucTransparent _ THEN (Returncode := C_RC_CmdIncompFStruc GOTO UPDATE_BINARY_Exit) FileOffset := (APDU.Cmd.P1 * 256) + APDU.Cmd.P2</pre>	Prüfe, ob bereits ein EF selektiert ist. Falls nein, Abbruch des Kommandos. Setze einen Teil des Dateibaums als Referenz für dieses Kommando. Prüfe, ob das selektierte EF die Dateistruktur „transparent“ hat. Berechne Offset zu den Daten in der Datei.
<pre>IF Status(File_Manager_CheckACUpdate) = _ C_AccessDenied) THEN (Returncode := C_RC_SecStateNotSatisfied GOTO UPDATE_BINARY_Exit)</pre>	Prüfe, ob der Sicherheitszustand erreicht ist, damit in das selektierte EF mit UPDATE BINARY geschrieben werden kann.
<pre>IF .EF[Ptr.CurrentWEF].TransparentDataSize) < _ (FileOffset + APDU.Cmd.Lc) THEN (Returncode := C_RC_WrongP1P2 GOTO UPDATE_BINARY_Exit) CALL Kernel_CopyByteString // Von: APDU.Cmd.Data[1 ... APDU.Cmd.Lc] // Nach: .EF[Ptr.CurrentWEF].TransparentData[x ... y] // x = FileOffset ; y = (FileOffset + APDU.Cmd.Lc) IF Status(Kernel_CopyByteString) = C_WriteError THEN (Returncode := C_RC_MemoryFailure GOTO UPDATE_BINARY_Exit) Returncode := C_RC_OK UPDATE_BINARY_Exit: END WITH RETURN</pre>	Prüfe, ob gewählter Offset und angeforderte Datenlänge (= Lc) zur Dateigröße passen. Kopiere die übergebenen Daten vom I/O-Empfangspuffer in die Datei. Falls dabei ein Fehler auftritt, brich ab und melde dies dem Terminal. Das Kommando wurde fehlerfrei abgearbeitet, da in allen anderen Fällen der Fehleraussprung benutzt wurde.

Kommando READ RECORD

Mit dem Kommando READ RECORD kann aus einem EF mit der Struktur „linear fixed“ ein Record gelesen werden. Die maximale Länge der zu lesenden Daten ist auf die maximale Recordgröße von 255 Byte beschränkt. Die Längenangabe muss entweder der Länge des adressierten Records entsprechen oder null betragen. Bei Länge null wird automatisch der gesamte Record ausgelesen. Alle Längenangaben müssen ein Vielfaches von einem Byte sein. Im Profil N von ISO/IEC 7816-4 ist die implizite Selektion von EFs mittels Short-FID nicht vorgesehen und deshalb hier auch nicht realisiert. READ RECORD fällt unter Case 2, d. h. bei der Kommando-APDU ist kein Datenteil vorhanden, jedoch bei der Antwort-APDU.

Ein Record in einem EF der Struktur „linear fixed“ kann auf drei unterschiedliche Arten adressiert werden. In Kommando READ RECORD kann die Nummer des gewünschten Records direkt übergeben werden. Ist dieser Record in der Datei vorhanden, dann erhält man den Inhalt in den Antwortdaten zurück, andernfalls eine entsprechende Fehlermeldung. Dieser Zugriff beeinflusst den Record-Zeiger nicht. Er kann nur mit den Kommandooptionen „first“, „last“, „next“ und „previous“ verändert werden und ist direkt nach der Neuselektion des EFs auf ungültig gesetzt. Wird bei einem als ungültig gekennzeichneten Record-Zeiger die Option „next“ oder „previous“ gewählt, dann wird der Record-Zeiger automatisch auf den ersten bzw. auf den letzten Record in der Datei gesetzt. Damit ist es beispielsweise möglich, ohne zusätzliche Kommandos direkt nach der Selektion eines EFs durch wiederholtes Senden eines READ RECORD-Kommandos mit der Option „next“ alle Records nacheinander auszulesen. Die dritte Art des Zugriffs ist die Benutzung der Option „current“. Damit wird der Record gelesen, den der Record-Zeiger aktuell indiziert. Ist der Record-Zeiger ungültig, so wird das Kommando mit einer entsprechenden Fehlermeldung abgebrochen.

Um Daten aus einem EF mit diesem Kommando zu lesen, muss die dazugehörige Zugriffsbedingung erfüllt sein, sonst wird das Kommando mit einer entsprechenden Fehlermeldung abgewiesen. Der im Gutfall als Antwort zurückgegebene Record ist nicht TLV-codiert, was laut ISO/IEC 7816-4 optional möglich wäre.

Tabelle 5.60 Small-OS: Codierung des Case-2-Kommandos READ RECORD

Datenelement	Codierung	Bemerkung
CLA	'00'	---
INS	'B2'	---
P1	X	Y = °0000 0100°, X = 0 Lies den aktuellen Record (<i>Ptr.CurrentRecord</i>). Y = °0000 0100°, X <> 0 Lies den Record mit der Nummer X.
P2	Y	Y = °0000 0100° Lies den Record mit der in P1 übergebenen Methode. X = 0, Y = °0000 0000° Lies den ersten (<i>first</i>) Record in der Datei. X = 0, Y = °0000 0001° Lies den letzten (<i>last</i>) Record in der Datei. X = 0, Y = °0000 0010° Lies den nächsten (<i>next</i>) Record in der Datei. X = 0, Y = °0000 0011° Lies den vorherigen (<i>previous</i>) Record in der Datei.
Le	Z	Z = 0: Lies alle Bytes bis zum Ende des Records. Z > 0: Z ist die Länge des Records

Tabelle 5.61 Small-OS: Codierung der Antwort auf das Kommando READ RECORD

Datenelement	Codierung	Bemerkung
DATA	...	Falls das Kommando korrekt ausgeführt wurde, befindet sich in diesem Datenelement der mit dem Kommando angeforderte Record.
SW1 SW2	'9000'	Returncode für den Fall des korrekt ausgeführten Kommandos.

Tabelle 5.62 Programmcode des Small-OS: Kommando READ RECORD nach ISO 7816-4 Profil N

READ_RECORD: IF LENGTH (APDU.Cmd) < 5 THEN (Returncode := C_RC_WrongLength RETURN) IF APDU.Cmd.P2.b8 ... b4 <> °00000° THEN (Returncode := C_RC_FktNotSupported GOTO READ_RECORD_Exit) IF Ptr.CurrentWEF = C_InvalidPointer THEN (Returncode := C_RC_CmdNotAllowed GOTO READ_RECORD_Exit) WITH DF[Ptr.CurrentDF]. IF .EF[Ptr.CurrentWEF].Structure <> C_EFStrucLinFix THEN (Returncode := C_RC_CmdIncompFStruc GOTO READ_RECORD_Exit)	Kommando nach ISO/IEC 7816-4, Profil N Prüfe, ob das Kommando als Case-2-Kommando gesendet wurde. Ist dies nicht der Fall, setze den entsprechenden Returncode und brich die Kommandobearbeitung ab. Prüfe, ob ein EF mit Short-FID selektiert werden soll. Prüfe, ob bereits ein EF selektiert ist. Setze einen Teil des Dateibaums als Referenz für dieses Kommando.
IF Status(File_Manager_CheckACRead) = C_AccessDenied __ THEN (Returncode := C_RC_SecStateNotSatisfied GOTO READ_RECORD_Exit)	Prüfe, ob der Sicherheitszustand erreicht ist, damit aus dem selektierten EF mit READ RECORD gelesen werden darf.
RecordNoToRead := 0 RecordLenToRead := 0 IF APDU.Cmd.P2.b3 ... b1 = °000° THEN (Ptr.CurrentRecord := 1 RecordNoToRead := Ptr.CurrentRecord) IF APDU.Cmd.P2.b3 ... b1 = °001° THEN (Ptr.CurrentRecord := .EF[Ptr.CurrentWEF].NoOfRecords RecordNoToRead := Ptr.CurrentRecord) IF APDU.Cmd.P2.b3 ... b1 = °010° THEN (IF Ptr.CurrentRecord = C_InvalidPointer THEN (Ptr.CurrentRecord := 1 RecordNoToRead := Ptr.CurrentRecord) ELSE (IF Ptr.CurrentRecord < .EF[Ptr.CurrentWEF]. __ NoOfRecords THEN (Ptr.CurrentRecord := Ptr.CurrentRecord + 1 RecordNoToRead := Ptr.CurrentRecord) ELSE (Returncode = C_RC_RecordNotFound GOTO READ_RECORD_Exit))	Initialisiere die Variablen für die Nummer des zu lesenden Records und dessen Länge. Falls die Option „lese first Record“ gewählt wurde, setze den Zeiger „aktueller Record“ auf den ersten Record in der Datei. Falls die Option „lese last Record“ gewählt wurde, setze den Zeiger „aktueller Record“ auf den letzten Record in der Datei. Falls die Option „lese next Record“ gewählt wurde, setze den Zeiger „aktueller Record“ in der Datei, sofern er nicht schon auf den letzten Record zeigt.

Tabelle 5.62 (Fortsetzung)

<pre> IF APDU.Cmd.P2.b3 ... b1 = °011° THEN (IF Ptr.CurrentRecord = C_InvalidPointer THEN (Ptr.CurrentRecord = .EF[Ptr.CurrentWEF]. _ _NoOfRecords RecordNoToRead := Ptr.CurrentRecord) ELSE (IF Ptr.CurrentRecord > 1 THEN (Ptr.CurrentRecord := Ptr.CurrentRecord - 1 RecordNoToRead := Ptr.CurrentRecord) ELSE (Returncode = C_RC_RecordNotFound GOTO READ_RECORD_Exit))) IF ((APDU.Cmd.P2.b3 ... b1 = °100°) AND _ (APDU.Cmd.P1 = 0)) THEN (IF Ptr.CurrentRecord <> C_InvalidPointer THEN (Returncode = C_RC_WrongP1P2 GOTO READ_RECORD_Exit) ELSE (RecordNoToRead := Ptr.CurrentRecord)) IF (APDU.Cmd.P2.b3 ... b1 = °100°) AND (P1>0) THEN (IF .EF[Ptr.CurrentWEF].NoOfRecords < APDU.Cmd.P1 _ THEN (Returncode = C_RC_WrongP1P2 GOTO READ_RECORD_Exit) ELSE (RecordNoToRead := APDU.Cmd.P1) IF APDU.Cmd.Le = '00' THEN (RecordLenToRead := .EF[Ptr.CurrentWEF]. _ Record[RecordNoToRead].Size) ELSE (RecordLenToRead := APDU.Cmd.Le) IF RecordLenToRead <> .EF[Ptr.CurrentWEF]. _ Record[RecordNoToRead].Size THEN (Returncode = C_RC_LcInconsistentP1P2 GOTO READ_RECORD_Exit) CALL Kernel_CopyByteString // Von: .EF[Ptr.CurrentWEF].Record[RecordNoToRead]. _ // Data[1 ... RecordLenToRead] // Nach: APDU.Rsp.Data[1 ... RecordLenToRead] Returncode = C_RC_OK READ_RECORD_Exit: END WITH RETURN </pre>	Falls die Option „lese previous Record“ gewählt wurde, setze den Zeiger auf den vorherigen Record in der Datei, sofern er nicht schon auf den ersten Record zeigt. Falls die Option „lese current Record“ gewählt wurde, prüfe, ob der entsprechende Pointer gültig ist, und setze im Gutfall die Kommando-interne Variable des aktuell zu lesenden Records. Falls die Option „direkte Adressierung eines Records mit P1“ gewählt wurde, prüfe, ob er in der Datei vorhanden ist. Prüfe, ob der ganze Record mit expliziter Längenangabe oder ohne explizite Längenangabe (d.h. Le = '00') gelesen werden soll. Prüfe, ob die angeforderte Datenlänge (= Le) zur Recordgröße passt. Kopiere die angeforderten Daten von der Datei in den I/O-Sendepuffer. Das Kommando wurde fehlerfrei abgearbeitet, da in allen anderen Fällen der Fehleraussprung benutzt wurde.

Kommando UPDATE RECORD

Mit dem Kommando UPDATE RECORD kann in ein EF mit der Struktur „linear fixed“ ein Record geschrieben werden. Die im Kommando an die Chipkarte übergebenen Daten dürfen nicht TLV-codiert sein, was laut ISO/IEC 7816-4 optional möglich wäre. Die ma-

ximale Länge der zu schreibenden Daten ist auf die maximale Recordgröße von 255 Byte beschränkt. Die Längenangabe der Daten muss exakt der Länge des adressierten Records entsprechen, und alle Längenangaben müssen ein Vielfaches von einem Byte sein. Im Profil N von ISO/IEC 7816-4 ist die implizite Selektion von EFs mittels Short-FID nicht vorgesehen und deshalb hier auch nicht realisiert. UPDATE RECORD fällt unter Case 3, was bedeutet, da bei der Kommando-APDU ein Datenteil vorhanden ist, jedoch nicht bei der Antwort-APDU.

Ein Record in einem EF der Struktur „linear fixed“ kann auf drei unterschiedliche Arten adressiert werden. In Kommando UPDATE RECORD kann die Nummer des gewünschten Records direkt übergeben werden. Ist dieser Record in der Datei vorhanden, erhält man den Inhalt in den Antwortdaten zurück, andernfalls eine entsprechende Fehlermeldung. Dieser Zugriff beeinflusst den Record-Zeiger nicht. Er kann nur mit den Kommandooptionen „first“, „last“, „next“ und „previous“ verändert werden und ist direkt nach der Neuselektion des EFs auf ungültig gesetzt.

Wird bei einem als ungültig gekennzeichneten Record-Zeiger die Option „next“ oder „previous“ gewählt, dann wird der Record-Zeiger automatisch auf den ersten bzw. auf den letzten Record in der Datei gesetzt. Somit ist es beispielsweise möglich, ohne zusätzliche Kommandos direkt nach der Selektion eines EFs durch wiederholtes Senden eines UPDATE RECORD-Kommandos mit der Option „next“ alle Records nacheinander zu schreiben. Die dritte Art des Zugriffs ist die Benutzung der Option „current“. Damit wird der Record geschrieben, auf den der Record-Zeiger aktuell zeigt. Ist der Record-Zeiger ungültig, so wird das Kommando mit einer entsprechenden Fehlermeldung abgebrochen.

Um Daten in ein EF mit diesem Kommando zu schreiben, muss die dazugehörige Zugriffsbedingung erfüllt sein, sonst wird das Kommando mit einer entsprechenden Fehlermeldung abgewiesen.

Tabelle 5.63 Small-OS: Codierung des Case-3-Kommandos UPDATE RECORD

Datenelement	Codierung	Bemerkung
CLA	'00'	---
INS	'DC'	---
P1	X	Y = °0000 0100°, X = 0 Schreibe den aktuellen Record (<i>Ptr.CurrentRecord</i>). Y = °0000 0100°, X > 0 Schreibe den Record mit der Nummer X.
P2	Y	Y = °0000 0100° Schreibe den Record mit der in P1 übergebenen Methode. X = 0, Y = °0000 0000° Schreibe den ersten (<i>first</i>) Record in der Datei. X = 0, Y = °0000 0001° Schreibe den letzten (<i>last</i>) Record in der Datei. X = 0, Y = °0000 0010° Schreibe den nächsten (<i>next</i>) Record in der Datei. X = 0, Y = °0000 0011° Schreibe den vorherigen (<i>previous</i>) Record in der Datei.
Lc	...	Anzahl der zu schreibenden Bytes.
DATA	...	Der zu schreibende Record.

Tabelle 5.64 Small-OS: Codierung der Antwort auf das Kommando UPDATE RECORD

Datenelement	Codierung	Bemerkung
SW1 SW2	'9000'	Returncode für den Fall des korrekt ausgeführten Kommandos.

Tabelle 5.65 Programmcode des Small-OS: Kommando UPDATE RECORD nach ISO 7816-4 Profil

UPDATE_RECORD:	Kommando nach ISO/IEC 7816-4, Profil N
<pre> IF LENGTH (APDU.Cmd) < 6 THEN (Returncode := C_RC_WrongLength RETURN) IF APDU.Cmd.P2.b8 ... b4 <> °00000° THEN (Returncode := C_RC_FktNotSupported GOTO UPDATE_RECORD_Exit) IF Ptr.CurrentWEF = C_InvalidPointer THEN (Returncode := C_RC_CmdNotAllowed GOTO UPDATE_RECORD_Exit) WITH DF[Ptr.CurrentDF]. IF .EF[Ptr.CurrentWEF].Structure = C_EFStrucLinFix THEN (Returncode := C_RC_CmdIncompFStruc GOTO UPDATE_RECORD_Exit) IF Status(File_Manager_CheckACUpdate) = _ C_AccessDenied) THEN (Returncode := C_RC_SecStateNotSatisfied GOTO UPDATE_RECORD_Exit) RecordNoToUpdate := 0 IF APDU.Cmd.P2.b3 ... b1 = °000° THEN (Ptr.CurrentRecord := 1 RecordNoToUpdate := Ptr.CurrentRecord) IF APDU.Cmd.P2.b3 ... b1 = °001° THEN (Ptr.CurrentRecord := .EF[Ptr.CurrentWEF].NoOfRecords RecordNoToUpdate := Ptr.CurrentRecord) IF APDU.Cmd.P2.b3 ... b1 = °010° THEN (IF Ptr.CurrentRecord = C_InvalidPointer THEN (Ptr.CurrentRecord := 1 RecordNoToUpdate := Ptr.CurrentRecord) ELSE (IF Ptr.CurrentRecord < .EF[Ptr.CurrentWEF]. _ NoOfRecords THEN (</pre>	Prüfe, ob das Kommando als Case-3-Kommando gesendet wurde. Ist dies nicht der Fall, setze den entsprechenden Returncode und brich die Kommandobearbeitung ab. Prüfe, ob ein EF mit Short-FID selektiert werden soll. Prüfe, ob bereits ein EF selektiert ist. Setze einen Teil des Dateibaums als Referenz für dieses Kommando. Prüfe, ob das selektierte EF die Dateistruktur „linear fixed“ hat. Prüfe, ob der Sicherheitszustand erreicht ist, damit in das selektierte EF mit UPDATE RECORD geschrieben werden kann. Initialisiere die Variable für die Nummer des zu schreibenden Records. Falls die Option „schreibe first Record“ gewählt wurde, setze den Zeiger „aktueller Record“ auf den ersten Record in der Datei. Falls die Option „schreibe last Record“ gewählt wurde, setze den Zeiger „aktueller Record“ auf den letzten Record in der Datei. Falls die Option „schreibe next Record“ gewählt wurde, setze den Zeiger „aktueller Record“ in der Datei, sofern er nicht schon auf den letzten Record zeigt.

Tabelle 5.65 (Fortsetzung)

<pre>Ptr.CurrentRecord := Ptr.CurrentRecord + 1 RecordNoToUpdate := Ptr.CurrentRecord) ELSE (Returncode = C_RC_RecordNotFound GOTO UPDATE_RECORD_Exit))) IF APDU.Cmd.P2.b3 ... b1 = °011° THEN (IF Ptr.CurrentRecord = C_InvalidPointer THEN (Ptr.CurrentRecord = .EF[Ptr.CurrentWEF]. _ _NoOfRecords) RecordNoToUpdate := Ptr.CurrentRecord) ELSE (IF Ptr.CurrentRecord > 1 THEN (Ptr.CurrentRecord := Ptr.CurrentRecord - 1) RecordNoToUpdate := Ptr.CurrentRecord) ELSE (Returncode = C_RC_RecordNotFound GOTO UPDATE_RECORD_Exit))) IF ((APDU.Cmd.P2.b3 ... b1 = °100°) AND _ (APDU.Cmd.P1 = 0)) THEN (IF Ptr.CurrentRecord <> C_InvalidPointer THEN (Returncode = C_RC_WrongP1P2 GOTO UPDATE_RECORD_Exit) ELSE (RecordNoToUpdate := Ptr.CurrentRecord)) IF (APDU.Cmd.P2.b3 ... b1 = °100°) AND (P1>0) THEN (IF .EF[Ptr.CurrentWEF].NoOfRecords < APDU.Cmd.P1 _ THEN (Returncode = C_RC_WrongP1P2 GOTO UPDATE_RECORD_Exit)</pre>	Falls die Option „schreibe previous Record“ gewählt wurde, setze den Zeiger „aktueller Record“ in der Datei, sofern er nicht schon auf den ersten Record zeigt. Falls die Option „schreibe current Record“ gewählt wurde, prüfe, ob der entsprechende Pointer gültig ist, und setze im Gutfall die Kommando-interne Variable des aktuell zu schreibenden Records. Falls die Option „direkte Adressierung eines Records mit P1“ gewählt wurde, prüfe, ob er in der Datei vorhanden ist.
<pre>IF .EF[Ptr.CurrentWEF].Record[Ptr.CurrentRecord].Size _ <> APDU.Cmd.Lc THEN (Returncode := C_RC_LcInconsitentP1P2 GOTO UPDATE_RECORD_Exit) CALL Kernel_CopyByteString // Von: APDU.Cmd.Data[1 ... APDU.Cmd.Lc] // Nach: .EF[Ptr.CurrentWEF].Record[RecordNoToUpdate]. _ // Data[1 ... APDU.Cmd.Lc] IF Status(Kernel_CopyByteString) = C_WriteError THEN (Returncode := C_RC_MemoryFailure GOTO UPDATE_RECORD_Exit) Returncode := C_RC_OK UPDATE_RECORD_Exit: END WITH RETURN</pre>	Prüfe, ob die Länge des übergebenen Records Datenlänge (= Lc) zur Recordgröße in der Datei passt. Kopiere die übergebenen Daten vom I/O-Empfangspuffer in die Datei. Falls dabei ein Fehler auftritt, brich ab, und melde dies dem Terminal. Das Kommando wurde fehlerfrei abgearbeitet, in allen anderen Fällen wurde der Fehleraussprung benutzt.

Kommando VERIFY

Das Kommando VERIFY wird benutzt, um ein an die Chipkarte übergebenes Geheimnis, z. B. eine PIN, mit einem gespeicherten Referenzwert zu vergleichen. Die Länge der PIN muss zwischen einem und acht Byte sein, wobei das Betriebssystem keinerlei Codierung des übergebenen Datenstrings prüft. So kann beispielsweise eine 4-stellige PIN (1234) als zwei Byte in BCD ('12' || '34') oder in vier Byte ASCII-codiert sein ("1" || "2" || "3" || "4" = '31' || '32' || '33' || '34'). VERIFY fällt unter Case 3, was bedeutet, dass bei der Kommando-APDU ein Datenteil vorhanden ist, nicht jedoch bei der Antwort-APDU.

Es können maximal zwei PINs (PIN Nummer 1 oder 2) adressiert werden, welche sich entweder im EF Key des MFs oder im EF Key des aktuell selektierten DFs befinden können. Eine PIN, die im EF Key des MFs gespeichert ist, benutzt man als einheitliche PIN für alle Anwendungen auf der Chipkarte. Ist eine PIN im EF Key eines DFs abgelegt, so kann diese nur für die jeweilige Anwendung verwendet werden, sie ist also eine anwendungsspezifische PIN.

Jede PIN ist mit einem Fehlbedienungsähler versehen, der bei einem positiven Vergleichsergebnis auf den Wert null zurückgesetzt wird und bei einem negativen Vergleichsergebnis um eins erhöht wird. Ist der Stand des Fehlbedienungsählers ungleich null, so wird die Zahl der noch möglichen PIN-Vergleiche in SW2 codiert zurückgegeben. Hat der Fehlbedienungsähler seinen Maximalwert erreicht, wird dies mit einem separaten Returncode angezeigt.

Da kein Kommando zum Zurücksetzen des Fehlbedienungsählers in Small-OS vorhanden ist, bedeutet ein auf dem Maximalwert stehender Fehlbedienungsähler, dass unwiderprüflich keine weiteren PIN-Vergleiche mehr möglich sind. Dies kann abhängig von der Anwendung bedeuten, dass die Chipkarte nicht mehr benutzt werden kann. Obwohl es bei vielen Chipkarten-Anwendungen üblich ist, kann die im EF Key befindliche PIN leider nicht mehr durch den Benutzer geändert werden. Dazu wäre ein eigenes Kommando notwendig (CHANGE REFERENCE DATA nach ISO/IEC 7816-8), das aber nach ISO/IEC 7816-4 nicht vorgesehen ist.

Die dargestellte Implementation weist aus Vereinfachungsgründen eine kleine Besonderheit auf. Der Fehlbedienungsähler befindet sich bekanntermaßen im EEPROM. Da jedoch EEPROM-Schreibvorgänge durch einen potenziell möglichen Schreibfehler nicht immer erfolgreich verlaufen müssen, muss konsequenterweise nach jedem Schreibvorgang geprüft werden, ob die Daten in das EEPROM korrekt geschrieben wurden. Im Fehlerfall wird dann ein entsprechender Returncode gesetzt. Im VERIFY-Kommando wird der Fehlbedienungsähler so häufig verändert, dass im Pseudocode keine entsprechende Abfrage eingefügt wurde, da sonst die grundlegenden Zusammenhänge nicht mehr klar sichtbar wären. Dies sollte man berücksichtigen, wenn man sich den Pseudocode genauer ansieht.

Das Kommando VERIFY ist natürlich prädestiniert für Angriffe auf die PIN. Die Implementation ist so ausgeführt, dass Angriffe über Analyse des Zeitverhaltens oder des Stromverbrauchs nicht möglich sind. Der Fehlbedienungsähler wird grundsätzlich immer vor dem Vergleich der übergebenen PIN mit der im EF Key gespeicherten Referenz PIN um eins erhöht. So ist sichergestellt, dass ein Abbruch der Stromzufuhr unmittelbar nach dem PIN-Vergleich beim Schreiben des Fehlbedienungsählers nicht dazu führt, dass der Fehlbedienungsähler nicht mehr erhöht wird und so ein Angreifer unbegrenzt viele PIN-Vergleiche durchführen kann.

Der eigentliche EEPROM-Schreibvorgang beim Erhöhen des Fehlbedienungsählers ist längst kein so trivialer Vorgang, wie man es sich im ersten Ansatz denkt. Der Fehlbedie-

nungszähler muss in seiner Codierung so aufgebaut sein, dass ein Abbruch während des Schreib- oder eventuell davor notwendigen Löschvorgangs nie dazu führen kann, dass der Fehlbedienungs­zähler auf seinen Ausgangswert null gesetzt ist. Die Betriebssystem-interne Codierung muss sich also an dem energieärmsten, d. h. dem sicheren, Zustand des EEPROMs orientieren. Dies bedeutet beispielsweise bei einem EEPROM mit dem sicheren Zustand null, dass der Ausgangswert des Fehlbedienungs­zählers nicht als null codiert sein darf. Wäre dies der Fall, könnte man durch geschicktes Abschalten der Versorgungsspannung beim Schreiben des Fehlbedienungs­zählers diesen wieder auf null zurücksetzen. Die PIN wäre dann innerhalb kurzer Zeit durch try-and-error ermittelbar, da der Fehlbedienungs­zähler seiner Funktion des Zählens von negativen PIN-Vergleichen nicht mehr nachkommen kann. Idealerweise kann man hier beim Schreiben des Fehlbedienungs­zählers auch die Methodik von atomaren Abläufen²⁵ verwenden.

Tabelle 5.66 Small-OS: Codierung des Case-3-Kommandos VERIFY

Datenelement	Codierung	Bemerkung
CLA	'00'	---
INS	'20'	---
P1	'00'	---
P2	Y	Z = °0 0001° · Z = °0 0010° Nummer der referenzierten PIN (1 oder 2) Y = °100Z ZZZZ° Benutze eine Referenz PIN, die im EF Key im aktuell selektierten Verzeichnis (d. h. MF oder DF) gespeichert ist (specific reference data).
Lc	...	Länge der übergebenen PIN.
DATA	...	Die übergebene PIN.

Tabelle 5.67 Small-OS: Codierung der Antwort auf das Kommando VERIFY

Datenelement	Codierung	Bemerkung
SW1 SW2	'9000'	Returncode für den Fall des korrekt ausgeführten Kommandos (d. h. der PIN-Vergleich war erfolgreich).

Tabelle 5.68 Programmcode des Small-OS: Kommando VERIFY nach ISO 7816-4 Profil N

VERIFY: IF LENGTH {APDU.Cmd} < 6 THEN (Returncode := C_RC_WrongLength RETURN) IF APDU.Cmd.P1 <> '00' THEN (Returncode := C_RC_WrongP1P2 GOTO VERIFY_Exit)	Kommando VERIFY nach ISO/IEC 7816-4, Profil N Prüfe, ob das Kommando als Case-3-Kommando gesendet wurde. Ist dies nicht der Fall, setze den entsprechenden Returncode, und brich die Kommandobearbeitung ab. Prüfe, ob P1 den erlaubten Wert hat (d. h. P1 muss '00' sein).

²⁵ Siehe auch Abschnitt 5.10 „Atomare Abläufe“.

Tabelle 5.68 (Fortsetzung)

<pre> IF ((APDU.Cmd.P2 < '01') OR (APDU.Cmd.P2 > '02')) THEN (Returncode := C_RC_WrongP1P2 GOTO VERIFY_Exit) IF ((APDU.Cmd.Lc <= 1) OR (APDU.Cmd.Lc >= 8)) THEN (Returncode := C_RC_LcInconsistentP1P2 GOTO VERIFY_Exit) </pre>	Prüfe, ob P2 einen der beiden erlaubten Werte der Referenznummer der PIN hat (d. h. P2 muss entweder 1 oder 2 sein). Prüfe, ob die übergebenen Daten (d. h. die PIN) sich innerhalb der erlaubten Länge bewegen (d. h. $1 \leq Lc \leq 8$).
<pre> IF Ptr.CurrentIEF.Key = C_InvalidPointer THEN (Returncode := C_RC_RefDataNotFound GOTO VERIFY_Exit) SEARCH (nach der PIN mit der angeforderten Referenznummer in DF[Ptr.CurrentDF].EF[Ptr.CurrentIEF.Key]) IF STATUS(SEARCH) = C_Found THEN (setze KeyRecord auf den Record mit der gefundenen PIN WITH DF[Ptr.CurrentDF].EF[Ptr.CurrentIEF.Key].) ELSE (Returncode := C_RC_RefDataNotFound GOTO VERIFY_Exit) </pre>	Prüfe, ob in dem aktuellen Verzeichnis ein EF Key vorhanden ist. Suche die angeforderte Referenznummer im EF Key. Falls eine PIN mit der angegebenen Referenznummer gefunden wurde, setze darauf den Zeiger für den aktuellen Schlüssel (d. h. die PIN).
<pre> IF .Record[KeyRecord].KeyPurpose <> C_CmdVERIFY THEN (Returncode := C_RC_CondOfUseNotSatisfied GOTO VERIFY_Exit) IF Ptr.CurrentDF = Ptr.MF THEN (// MF ist selektiert IF ((.Record[KeyRecord].EntryState _ <> SecurityState.MF) THEN (Returncode := C_RC_SecStateNotSatisfied GOTO VERIFY_Exit)) ELSE (// DF ist selektiert IF .Record[KeyRecord].EntryState <> SecurityState.DF _ THEN (Returncode := C_RC_SecStateNotSatisfied GOTO VERIFY_Exit) IF .Record[KeyRecord].RCntr >= _ .Record[KeyRecord].RCntrMax THEN (Returncode := C_RC_AuthMethodBlocked GOTO VERIFY_Exit) IF .Record[KeyRecord].Lc <> .Record[KeyRecord].KeySize THEN (Returncode := C_RC_LcInconsistentP1P2 GOTO VERIFY_Exit) </pre>	Prüfe, ob die ausgewählten Daten die Benutzung mit dem Kommando VERIFY zulassen. Prüfe, ob der aktuelle Sicherheitszustand im MF bzw. DF die Benutzung mit dem Kommando VERIFY zulässt. Es muss abhängig vom aktuell selektierten Verzeichnis (d. h. MF oder DF) der im Schlüssel-Record geforderte Sicherheitszustand erreicht sein. Prüfe, ob der Fehlbedienungsähler abgelaufen ist. Prüfe, ob die übergebene PIN die gleiche Länge wie die gespeicherte Referenz-PIN hat.
<pre> .Record[KeyRecord].RCntr := .Record[KeyRecord].RCntr + 1 CALL Kernel_CompareByteString // Daten 1: APDU.Cmd.Data[1 ... APDU.Cmd.Lc] // Daten 2: .Record[KeyRecord].KeyData[1 ... APDU.Cmd.Lc] </pre>	Erhöhe vor dem eigentlichen PIN-Vergleich prophylaktisch den Fehlbedienungsähler. Damit kann ein eventueller Angriff über Zeit- oder Stromanalyse des PIN-Vergleichs bereits im Vorfeld abgewehrt werden. Vergleiche die übergebene PIN mit der gespeicherten Referenz-PIN.

Tabelle 5.68 (Fortsetzung)

<pre>IF STATUS(Kernel_CompareByteString) = C_Equal THEN (.Record[KeyRecord].RCntr := 0 IF Ptr.CurrentDF = Ptr.MF THEN (// MF ist selektiert SecurityState.MF := .Record[KeyRecord].ResultState.OK) ELSE (// DF ist selektiert SecurityState.DF := .Record[KeyRecord].ResultState.OK) IF STATUS(Kernel_CompareByteString) = C_NotEqual _ THEN (IF Ptr.CurrentDF = Ptr.MF THEN (// MF ist selektiert SecurityState.MF := _ .Record[KeyRecord].ResultState.NOK) ELSE (// DF ist selektiert SecurityState.DF := _ .Record[KeyRecord].ResultState.NOK)) IF .Record[KeyRecord].RCntr = _ .Record[KeyRecord].RCntrMaxValue THEN (Returncode := C_RC_AuthMethodBlocked ELSE (Returncode := C_RC_CounterX (.Record[KeyRecord].RCntrMax - .Record[KeyRecord].RCntr))) GOTO VERIFY_Exit) Returncode := C_RC_OK VERIFY_Exit: END WITH RETURN</pre>	Falls die übergebene PIN der gespeicherten Referenz PIN entspricht, setze den Fehlbedienungs­zähler auf null Fehlversuche und setze den Sicherheits­zustand für erfolgrei­che PIN-Prüfung. Falls die übergebene PIN nicht der gespei­cherten entspricht, wurde der Fehlbedie­nungs­zähler bereits vor dem PIN-Vergleich um eins erhöht. Setze den aus dem erfolglosen PIN-Vergleich resultie­renden Zustand. Falls der Fehlbedienungs­zähler seinen Maxi­malwert nicht erreicht hat, setze SW2 auf die Anzahl der noch verbleibenden Fehlver­suche, andernfalls zeige im Returncode, dass keine weiteren PIN-Überprüfungen mehr möglich sind. Das Kommando wurde fehlerfrei abgearbeitet, da in allen anderen Fäl­len der Fehler­aussprung benutzt wurde.

Kommando INTERNAL AUTHENTICATE

Das Kommando INTERNAL AUTHENTICATE wird benutzt, um die Chipkarte durch ein Challenge-Response-Verfahren zu authentisieren. Dazu wird an die Chipkarte eine acht Byte lange Zufallszahl übergeben und diese mittels des DES-Algorithmus verschlüsselt. Die Nummer des dazu verwendeten Schlüssels muss in Parameter P2 übergeben werden. Ebenfalls muss in P2 angegeben werden, ob sich der Schlüssel im EF Key des MFs oder des aktuell selektierten DFs befindet. INTERNAL AUTHENTICATE fällt unter Case 4, was bedeutet, dass sowohl bei der Kommando-APDU als auch bei der Antwort-APDU ein Datenteil vorhanden ist.

Die ISO/IEC 7816-4 spezifiziert wenig Parameter für die Kommandos zur Authentisie­rung. So ist etwa der kryptografische Algorithmus dort nicht festgelegt. Bei Small-OS wurde als voreingestellter Verschlüsselungsalgorithmus der DES gewählt, und ergän­

zend zu Profil N der ISO/IEC 7816-4 wird eine 5 Bit lange Schlüsselnummer mit übergeben.

INTERNAL AUTHENTICATE bei Small-OS ließe sich im Prinzip zur Verschlüsselung von Daten benutzen, da 8 Byte Klartext in 8 Byte Schlüsseltext mit einem wählbaren Schlüssel verschlüsselt werden. Dies würde dazu führen, dass so eine Karte in beinahe allen Ländern unter verschärftes Exportrecht fällt und eine wochen- oder zum Teil sogar monatelange Prozedur zur Ausfuhr dieser Chipkarten notwendig wäre. Deshalb wird in vielen echten Chipkarten-Betriebssystemen das INTERNAL AUTHENTICATE so implementiert, dass eine direkte Verschlüsselung von Daten nicht mehr möglich ist. Auf diese Weise kann man die Exportbeschränkungen umgehen.

Aus kryptografischer Sicht ist die direkte Verschlüsselung eines Klartextblocks in einen Schlüsseltextblock ebenfalls bedenklich, da man damit ein Klartext-Schlüsseltext-Paar für einen Brute-force-Angriff erzeugen kann. Da eine wiederholte unrechtmäßige Nutzung von INTERNAL AUTHENTICATE nicht durch einen Fehlbedienungszähler abgesichert werden kann, wäre diese Implementation ebenfalls sehr anfällig gegenüber einer Differenzial Fault Analyse.²⁶ Der am einfachsten zu realisierende Angriff wäre eine Zeitanalyse²⁷ (*timing attack*) bei der Berechnung des DES, welcher deshalb in jedem Fall rauschfrei sein muss, damit nicht aufgrund von Messungen der Berechnungszeit der Schlüssel ermittelt werden kann.

Aus allen oben dargestellten Gründen wird in der Praxis meist der Eingangswert durch eine intern in der Karte generierte Zufallszahl und die einzigartige Kartenummer ergänzt, verschlüsselt und der erzeugte Schlüsseltext mit den ergänzten Daten im Klartext wieder an das Terminal zurückgegeben. Damit kann man dieses Kommando nicht mehr zur Datenverschlüsselung verwenden (Exportproblematik), und es wird nicht immer der gleiche Wert verschlüsselt, was eine der Grundlagen des Schutzes sowohl gegen eine differenzielle Fehleranalyse (DFA) als auch gegen eine differenzielle Leistungsanalyse (DPA) ist.²⁸ All diese Maßnahmen zeigen relativ drastisch, dass ein im ersten Augenschein so simples Kommando wie INTERNAL AUTHENTICATE ein erhebliches Know-how bei der Spezifikation und Implementation erfordert, damit die Schlüssel einer Chipkartenanwendung nicht angreifbar werden.

Tabelle 5.69 Small-OS: Codierung des Case-4-Kommandos INTERNAL AUTHENTICATE

Datenelement	Codierung	Bemerkung
CLA	'00'	---
INS	'88'	---
P1	'00'	---
P2	Y	°Z ZZZZ° Nummer des referenzierten Schlüssels (1 ... 31) Y = °100Z ZZZZ° Benutze einen Schlüssel aus dem EF Key im aktuell selektierten Verzeichnis (d. h. MF oder DF) (<i>specific reference data</i>).
Lc	8	Länge der übergebenen Zufallszahl.
DATA	...	Die übergebene Zufallszahl.
Le	8	Länge der zurückzugebenden Zufallszahl.

²⁶ Siehe auch Abschnitt 8.2.4 „Angriffe und Abwehrmaßnahmen während der Kartenbenutzung“.

²⁷ Siehe auch Abschnitt 8.2.4.2 „Angriffe auf der logischen Ebene“.

²⁸ Siehe auch Abschnitt 8.2.4.2 „Angriffe auf der logischen Ebene“.

Tabelle 5.70 Small-OS: Codierung der Antwort auf das Kommando INTERNAL AUTHENTICATE

Datenelement	Codierung	Bemerkung
DATA	...	Falls das Kommando korrekt ausgeführt wurde, befindet sich in diesem Datenelement die mit dem im Kommando referenzierten Schlüssel verschlüsselte Zufallszahl.
SW1 SW2	'9000'	Returncode für den Fall des korrekt ausgeführten Kommandos.

Tabelle 5.71 Programmcode des Small-OS: Kommando INTERNAL AUTHENTICATE nach ISO 7816-4 Profil N mit der Erweiterung globale und spezifische Referenzdaten (d. h. wahlweise Referenz auf MF oder DF)

INTERNAL_AUTHENTICATE: <pre> IF APDU.Cmd.P1 <> '00' THEN (Returncode := C_RC_WrongP1P2 GOTO INTERNAL_AUTHENTICATE_Exit) IF APDU.Cmd.Lc <> 8 THEN (Returncode := C_RC_WrongLength GOTO INTERNAL_AUTHENTICATE_Exit) </pre>	Kommando INTERNAL AUTHENTICATE nach ISO/IEC 7816-4, Profil N mit Erweiterung Prüfe, ob P1 den erlaubten Wert hat (d.h. P1 muss '00' sein). Prüfe, ob die übergebenen Daten (d. h. die Zufallszahl) die erlaubte Länge von 8 Byte haben.
<pre> IF Ptr.CurrentIEF.Key = C_InvalidPointer THEN (Returncode := C_RC_RefDataNotFound GOTO INTERNAL_AUTHENTICATE_Exit) IF APDU.Cmd.P2.b5 ... b1 = °00000° THEN (Returncode := C_RC_WrongP1P2 GOTO INTERNAL_AUTHENTICATE_Exit) ELSE (KeyNumber := APDU.Cmd.P1.b5 ... b1) SEARCH (nach dem Schlüssel mit der KeyNumber in _ DF[Ptr.CurrentDF].EF[Ptr.CurrentIEF.Key]) IF STATUS(SEARCH) = C_Found THEN (setze KeyRecord auf den Record mit dem gefundenen Schlüssel WITH DF[Ptr.CurrentDF].EF[Ptr.CurrentIEF.Key].) ELSE (Returncode := C_RC_RefDataNotFound GOTO INTERNAL_AUTHENTICATE_Exit) IF .Record[KeyRecord].KeyPurpose <> _ C_CmdINTAUTH THEN (Returncode := C_RC_CondOfUseNotSatisfied GOTO INTERNAL_AUTHENTICATE_Exit) IF Ptr.CurrentDF = Ptr.MF THEN (// MF ist selektiert IF .Record[KeyRecord].EntryState <> SecurityState.MF _ THEN (Returncode := C_RC_SecStateNotSatisfied GOTO INTERNAL_AUTHENTICATE_Exit)) ELSE (// DF ist selektiert IF .Record[KeyRecord].EntryState <> SecurityState.DF _ THEN (Returncode := C_RC_SecStateNotSatisfied GOTO INTERNAL_AUTHENTICATE_Exit) </pre>	Prüfe, ob in dem aktuellen Verzeichnis ein EF Key vorhanden ist. Ermittle aus P1 die übergebene Nummer des zu verwendenden Schlüssels. Suche die in P2 angeforderte Referenznummer im EF Key. Falls ein Schlüssel mit der angegebenen Referenznummer gefunden wurde, setze darauf den Zeiger für den aktuellen Schlüssel. Prüfe, ob der ausgewählte Schlüssel die Benutzung mit dem Kommando INTERNAL AUTHENTICATE zulässt. Abhängig vom aktuell selektierten Verzeichnis (d. h. MF oder DF) muss der im Schlüssel-Record geforderte Sicherheitszustand erreicht sein.

Tabelle 5.71 (Fortsetzung)

<pre>CALL Kernel_DES_Encrypt // Klartext: APDU.Cmd.Data[1 ... 8] // Schlüssel: .Record[KeyRecord].KeyData[1 ... 8] // Schlüsseltext: wird in APDU.Rsp.Data[1 ... 8] abge- legt Returncode := C_RC_OK INTERNAL_AUTHENTICATE_Exit: END WITH RETURN</pre>	Verschlüssele die übergebenen Daten (d. h. die Zufallszahl) mit dem referenzierten Schlüssel und lege das Ergebnis in den Sendepuffer. Das Kommando wurde fehlerfrei abgearbeitet, da in allen anderen Fällen der Fehleraussprung benutzt wurde.

Beispiel für eine einfache Anwendung

Aufbau und Inhalt der Variablen im EEPROM-Speicher bei Small-OS soll die folgende einfache Chipkarten-Anwendung verdeutlichen, deren Funktionalität in wenigen Worten beschrieben ist: Es soll eine 50 Byte große Datei erzeugt werden, deren Inhalt immer gelesen werden kann und nach erfolgreicher Prüfung der PIN '1234' auch überschrieben werden darf. Die PIN hat die Referenznummer eins, und es sind maximal drei Fehlversuche bei der PIN-Eingabe gestattet. Das EF mit den Daten befindet sich unter einem eigenen DF. Alle Dateinamen (DF Name, FID) können frei gewählt werden.

In Tabelle 5.64 ist aufgeführt, wie die geforderte Funktionalität durch Setzen der entsprechenden Werte in den Strukturen für den Dateibaum realisiert wurde. Um die geforderten Zugriffsbedingungen zu realisieren, wurde der in Bild 5.50 gezeigte Zustandsautomat definiert. Nach dem Reset wird von Small-OS automatisch der Sicherheitszustand null im DF gesetzt. In diesem Zustand ist das EF mit den Daten lesbar, jedoch nicht schreibbar, da hierzu Zustand eins notwendig ist. Führt man nun ein erfolgreiches VERIFY mit der richtigen PIN durch, dann wird im DF der Sicherheitszustand eins gesetzt („...ResultState.OK“) und die Datei kann geschrieben werden. Verläuft die PIN-Prüfung nicht erfolgreich, so wird im DF der Sicherheitszustand null gesetzt („...ResultState.NOK“).

Zum Lesen bzw. Schreiben werden die Kommandos READ BINARY und UPDATE BINARY verwendet. Es kann sowohl die ganze Datei als auch nur ein Teil davon gelesen bzw. geschrieben werden. Ausschlaggebend ist lediglich die Erreichung des betreffenden Sicherheitszustandes im DF.

Das Beispiel zeigt recht deutlich zwei Einschränkungen von Small-OS, die nur deshalb zustande kamen, weil der Pseudocode nicht noch umfangreicher werden sollte, als er es schon ist. Wenn der Sicherheitszustand eins erreicht ist, kann die Datei nicht mehr gelesen werden, da dies nur im Zustand null erlaubt ist. Abhilfe könnte dadurch geschaffen werden, dass man neben einem Vergleich auf „gleich“ auch die Option eines Vergleichs auf „größer-gleich“ in Small-OS integriert. Die andere Lösungsmöglichkeit wäre, die Dateizugriffsbedingungen nicht nur auf eine Vergleichsabfrage pro Operation (d. h. Lesen, Schreiben) zu beschränken, sondern auf mehrere pro Operation. Selbst wenn dann nur ein Vergleich auf Gleichheit möglich wäre, könnte man beispielsweise das Lesen in Zustand Null und zusätzlich auch in Zustand Eins erlauben.

Mehrfache Zugriffsbedingungen für eine Zugriffsoperation ließen sich ebenso einfach in Small-OS integrieren wie Abfragen auf größer oder größer-gleich. Der Pseudocode würde nur etwas umfangreicher und ebenso der Programmcodeumfang für eine echte Implementierung. Bei echten Chipkarten-Betriebssystemen könnten diese Erweiterungen leicht dazu führen, dass der zur Verfügung stehende Speicher (ROM bzw. EEPROM) auf dem Mikrocontroller nicht mehr ausreicht. Diese strikten Speicherplatzbeschränkungen wirken sich in der Praxis oft dahingehend aus, dass manche nützliche Funktion nicht implementiert werden kann.

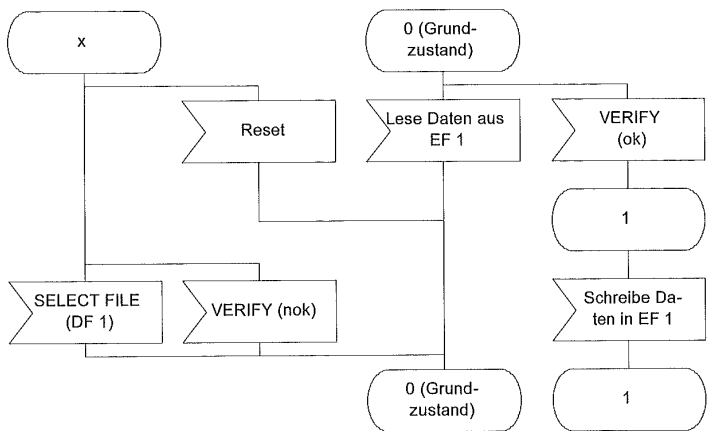


Bild 5.66 Das Zustandsdiagramm für die Datei mit der FID = '0001' (d.h. EF 1). Das Lesen und Schreiben auf das EF bezieht sich auf ein erfolgreiches READ BINARY bzw. UPDATE BINARY-Kommando an die Chipkarte. Der Zustand „x“ steht für beliebige Zustände.

Tabelle 5.72 Beispiel für die Belegung der Variablen der Dateiverwaltung bei einer einfachen Beispielanwendung. In dieser gibt es ein DF direkt unter dem MF mit einem EF. Dieses hat die Struktur „transparent“ und ist 50 Byte groß. Der Inhalt dieses EFs kann jederzeit gelesen werden, lässt sich jedoch nur nach PIN-Eingabe verändern.

Variablenname	Beschreibung, Inhalt und Größe
Datenstrukturen für das MF	
DF[1].FID := '3F00'	Die normierte FID des MFs.
Datenstrukturen für das DF	
DF[2].FID := 'DF01'	Die FID des DF ist 'DF01' (frei gewählt).
DF[2].DFName := 'D276' '000060'	Der DF Name wird hier mit dem registrierten AID von Wolfgang Rankl belegt.
DF[2].LenDFName := 5	Die Länge von DF Name ist 5 Byte.
Datenstrukturen des EF	
DF[2].EF[1].FID := '0001'	Die FID des EFs ist '0001' (frei gewählt).
DF[2].EF[1].Structure := C_EFStrucTransp	Die Struktur des EFs ist transparent.
DF[2].EF[1].AccessCondition.Read := 0	Der notwendige Zustand, damit das Kommando READ auf das EF erlaubt ist. Die Zugriffsbedingung ist auf 0 gesetzt, d. h. das Lesen der Datei ist immer erlaubt.

Tabell 5.72 (Fortsetzung)

<i>DF[2].EF[1].AccessCondition.Update := 1</i>	Der notwendige Zustand, damit das Kommando UP-DATE auf das EF erlaubt ist. Die Zugriffsbedingung ist auf 1 gesetzt, d. h. das Ändern der Datei ist immer nur nach einer erfolgreichen PIN-Eingabe erlaubt.
<i>DF[2].EF[1].TransparentDataSize := 50</i>	Die Größe der transparenten Datei ist 50 Byte.
Datenstrukturen des EF Key	
<i>DF[2].EF[2].Record[x].KeyData := '1234'</i>	Die PIN hat den (hexadezimalen) Wert '1234'.
<i>DF[2].EF[2].Record[x].KeySize := 2</i>	Die Länge der PIN beträgt 2 Byte.
<i>DF[2].EF[2].Record[x].KeyNo := 1</i>	Die PIN hat die Referenz-Nummer 1.
<i>DF[2].EF[2].Record[x].RCntr := 0</i>	Der Fehlbedienungs-zähler hat den Ausgangswert 0.
<i>DF[2].EF[2].Record[x].RCntrMaxValue := 3</i>	Der Maximalwert des Fehlbedienungs-zählers ist 3, d. h. der Benutzer hat bei der PIN-Eingabe maximal drei Fehlversuche.
<i>DF[2].EF[2].Record[x].KeyPurpose := C_CmdVERIFY</i>	Der Dateninhalt des EF Key Records darf nur für PIN-Prüfungen mit dem Kommando VERIFY verwendet werden.
<i>DF[2].EF[2].Record[x].EntryState := 0</i>	Die PIN-Prüfung darf nur im Zustand 0 durchgeführt werden.
<i>DF[2].EF[2].Record[x].ResultState.OK := 1</i>	Ist der PIN-Vergleich erfolgreich, wird der Zustand 1 im aktuellen DF gesetzt.
<i>DF[2].EF[2].Record[x].ResultState.NOK := 0</i>	Ist der PIN-Vergleich nicht erfolgreich, wird der Zustand 0 im aktuellen DF gesetzt.

Bei der beschriebenen Anwendung kann nach einer erfolgreich durchgeführten PIN-Überprüfung der Zustand eins nicht mehr verlassen werden, und die Datei würde sich aufgrund der Zugriffsbedingungen nicht mehr lesen lassen. Der Zustand null kann entweder durch eine nicht erfolgreiche PIN-Überprüfung oder durch einen Reset der Chipkarte erreicht werden. Dies ist seitens eines sauberen Applikationsdesigns eine ziemlich unglückliche Lösung. Man kann sich aber mit einem einfachen Trick behelfen, um wieder in den Ausgangszustand null zu kommen. Die Selektion eines DFs führt bekanntermaßen zum Zurücksetzen des aktuellen Sicherheitszustands für das DF. Deshalb führt man bei Bedarf eine nochmalige Selektion des DFs durch, und der Sicherheitszustand wird automatisch von eins wieder auf null zurückgesetzt.

Anwendungen wie die hier gezeigte wurden noch bis vor kurzem händisch in Assembler in der gezeigten Form durchgeführt. Mittlerweile gibt es aber für fast alle am Markt befindlichen Chipkarten-Betriebssysteme so genannte Applikationsgeneratoren, mit denen auf einer grafischen Benutzeroberfläche Dateien und Zugriffsbedingungen auf einem PC erstellt werden können. Ein ähnlicher Prozess läuft auch in The Smart Card Simulator ab, wenn man eine neue Datei oder Anwendung erstellt. Sind die für eine Anwendung notwendigen Dateien erzeugt, kann man diese Daten mit den Applikationsgeneratoren auf eine Chipkarte laden und erste Versuche mit der neuen Anwendung durchführen.

Die Beispielanwendung lässt sich auch nutzbringend für die Darstellung einiger interessanter Angriffe verwenden. Diese sind zwar eher theoretischer Natur, da sie einen großen Aufwand an technischer Ausrüstung erfordern, aber dennoch zeigen sie einige bemerk-

kenswerte Ansätze. Die Voraussetzung dafür ist, dass man Speicherinhalte im EEPROM gezielt verändern kann, was technisch darauf hinausläuft, den Ladungsinhalt des Floating Gate von einzelnen EEPROM-Zellen zu manipulieren. In Kapitel 8 Sicherheitstechnik wird auf die dazu notwendige Technik näher eingegangen. An diesen Stellen seien nur die Auswirkungen davon aufgezeigt.

Wäre es möglich, denjenigen Zeiger im Dateibaum, der normalerweise auf die Daten des EFs zeigt, gezielt zu ändern, so könnte man ihn dahingehend manipulieren, dass er auf den Dateninhalt des EF Key zeigt. Im Zustand null ist das EF immer lesbar, nur würde dann statt des eigentlichen Dateninhalts von 50 Byte mit READ BINARY die PIN ausgelesen werden. Allerdings ist für diesen Angriff das Wissen um die exakte Adresse des Datenteils des EF Key erforderlich. Diese herauszufinden erfordert aber sehr viel Insiderkenntnisse. Einfacher wäre es da schon, wenn man die Variable „DF[2].EF[1].-TransparentDataSize“ ändern könnte. Erhöht man den Wert dieser Variablen beispielsweise auf einen hohen Wert, kann mit READ BINARY entsprechend der neuen Dateigröße über die ursprüngliche Datei hinaus adressiert und damit gelesen werden. Befindet sich das EF Key im Speicher nach dem EF 1, würde sich sowohl der EF Key Header als auch der Datenteil ohne weiteres auslesen lassen.

Manipulationen im EEPROM-Speicher könnten auch für ein wiederholtes Rücksetzen des PIN-Fehlbedienungszählers genutzt werden, sodass sich mit try-and-error innerhalb erträglicher Zeit die PIN ermitteln ließe. Einfacher wäre sogar noch, die PIN selber auf einen bekannten Wert zu setzen.

Die Beispiele zeigen ziemlich deutlich, dass die Sicherheit einer Chipkarte bei einer Manipulationsmöglichkeit des EEPROMs vollständig zusammenbrechen würde. Es würde überhaupt keine Rolle spielen, ob EEPROM-Inhalte nur überschrieben, aber nicht gelesen werden könnten. In jedem Falle könnten die PIN und die Schlüssel der Chipkarte ermittelt werden. Das einzige Wissen, das man noch benötigt, sind die exakten Adressen im Speicher, an denen die Manipulationen vorgenommen werden müssten. Eventuell vorhandene Prüfsummen über die Dateiheder würden die notwendigen Eingriffe ins EEPROM nur komplizierter machen, aber letzten Endes nicht verhindern können. Allerdings ist es zum heutigen Zeitpunkt technisch nicht möglich, an beliebigen Adressen im EEPROM bitgenaue Änderungen vorzunehmen. Die vorstehenden Ansätze haben deshalb mehr den Charakter von theoretisch interessanten Angriffen als von in der Realität gefährlichen Attacken. Sollten jedoch in der Zukunft die oben vorausgesetzten Manipulationen des EEPROMs durchführbar werden, wurde hier deutlich und unmissverständlich das entstandene Gefährdungspotenzial aufgezeigt.

6 Datenübertragung zur Chipkarte

6.1	Physikalische Übertragungsschicht	379
6.2	Answer to Reset – ATR	384
6.3	Protocol Parameter Selection – PPS	399
6.4	Übertragungsprotokolle	403
6.4.1	Synchrone Datentübertragung	404
6.4.2	Übertragungsprotokoll T = 0.	410
6.4.3	Übertragungsprotokoll T = 1.	416
6.4.4	Übertragungsprotokoll T = 14 (Deutschland).	427
6.4.5	Übertragungsprotokoll USB	428
6.4.6	Vergleich der asynchronen Übertragungsprotokolle	428
6.5	Struktur der Nachrichten – APDUs	429
6.5.1	Struktur der Kommando-APDUs	430
6.5.2	Struktur der Antwort-APDUs	432
6.6	Sicherung der Datenübertragung	433
6.6.1	Das Authentic-Verfahren	436
6.6.2	Das Combined-Verfahren	438
6.6.3	Sendefolgezähler (<i>send sequence counter</i>).	440
6.7	Logische Kanäle (<i>logical channels</i>).	441

Voraussetzung für alle Interaktionen zwischen Chipkarte und Terminal ist die Möglichkeit der Kommunikation miteinander. Dafür steht aber 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 zurzeit in der Chipkartenwelt noch nicht verwirklicht. Da jedoch die meisten 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 ohne weiteres möglich. Mittelfristig wird sie mit Sicherheit in Hardware und Betriebssystemen implementiert sein.

Die Kommunikation mit der Chipkarte wird immer vom Terminal angestoßen, die Karte reagiert damit 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 Kommandos¹, bei dem die Chipkarte Kommandos an das Terminal senden kann, baut auf der üblichen Zuordnung von Master und Slave auf.

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

¹ Siehe auch Abschnitt 13.2.4 „Die SIM“.

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. Dieses Ping-Pong-Spiel von Kommando und Antwort setzt sich dann bis zum Deaktivieren der Chipkarte fort.

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.

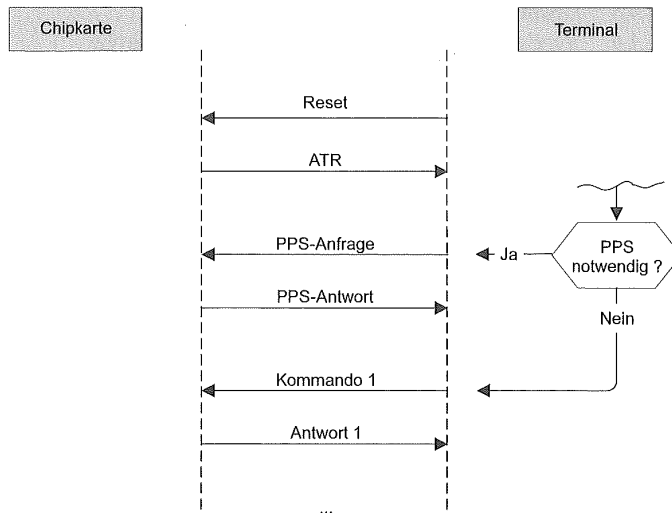


Bild 6.1 Answer to Reset (ATR), Protocol Parameter Selection (PPS) und das erste Kommando-Antwort-Paar bei der Datenübertragung zwischen Chipkarte und Terminal.

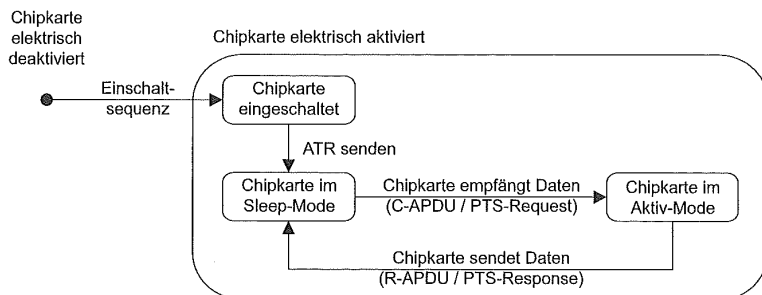


Bild 6.2 Die generellen Zustände einer Chipkarte beim Anschalten und mit der Kommunikation mit dem Terminal.

² Siehe auch Abschnitt 3.3.6 „An-/Abschaltsequenz“.

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

Funktional gesehen kann man die Variationsmöglichkeiten nur als ein Angebot von Varianten sehen, aus dem man für die Anwendung bzw. Chipkarte das Optimale herausucht. 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.

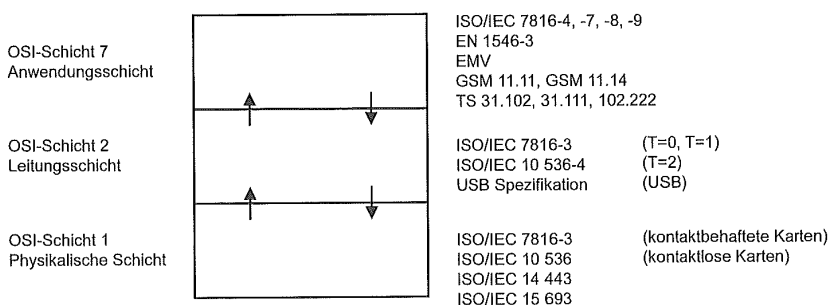


Bild 6.3 Das OSI-Modell der Kommunikation zwischen Terminal und Chipkarte

6.1 Physikalische Übertragungsschicht

Die physikalische Übertragungsschicht ist in der internationalen Norm für Chipkarten ISO/IEC 7816-3 in allen ihren Rahmenparametern festgelegt. Dies ist die grundlegende Norm für alle Aspekte der Kommunikation auf physikalischer Ebene.

Der gesamte Datenaustausch mit der Chipkarte findet auf digitalem Weg statt, d. h. nur unter Benutzung der logischen Werte 0 und 1. Die dafür verwendeten Spannungswerte liegen dabei auf den entsprechenden Werten der Spannungsversorgung, also bei + 5 V, + 3 V und + 1,8 V und bei 0 Volt als Referenz. Ob nun der Low- oder High-Pegel die logische „1“ symbolisiert, ist frei definierbar und wird von der Chipkarte im ersten Byte des ATR angezeigt. Dabei entspricht „direct convention“ einer logischen „1“ auf dem + 1,8/3/5-Volt-Pegel, „inverse convention“ bedeutet, dass der + 1,8/3/5-Volt-Pegel die logische „0“ symbolisiert. Im Ruhezustand, d. h. wenn keine Daten übertragen werden, befindet sich die I/O-Leitung in beiden Fällen immer auf High-Pegel.

Die Kommunikation zwischen einer Chipkarte und der äußeren Welt findet in serieller Form statt. Die vom Prozessor byteweise verarbeiteten Daten müssen deshalb in einen bitseriellen Datenstrom umgewandelt werden. Ein Byte wird in seine acht Einzelbits aufgetrennt, und diese werden dann nacheinander über die Leitung gesendet. Die Reihenfolge ist dabei ebenfalls von der convention abhängig. Bei direct convention ist das erste Datenbit nach dem Startbit das niederwertigste im Byte. Im Falle der inverse convention wird das höherwertigste Bit im Byte direkt nach dem Startbit gesendet.

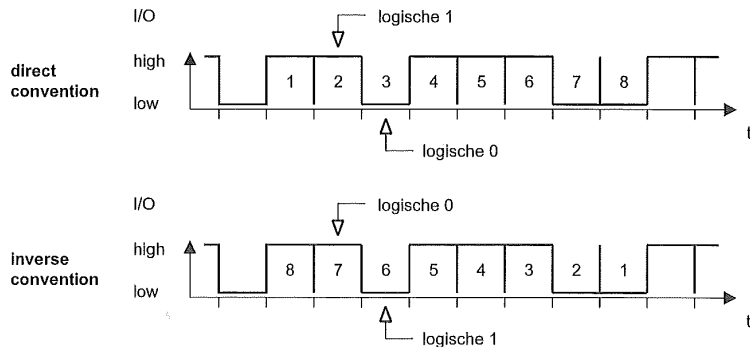


Bild 6.4 Die convention bei der Datentübertragung. Die obere Zeichnung zeigt die Datentübertragung in „direct convention“, während die untere Zeichnung die Datentübertragung in „inverse convention“ darstellt.

Die Datenübertragung zwischen Chipkarte und Terminal läuft asynchron ab, was bedeutet, dass jedes übertragene Byte mit zusätzlichen Synchronisationsbits ausgestattet werden muss. Den Anfang eines jeden seriell übertragenen Bytes ergänzt man dabei mit einem Startbit, das dem Empfänger den Sendebeginn signalisiert. Am Ende jedes Bytes fügt der Sender noch ein Paritätsbit zur Fehlererkennung sowie ein oder zwei Stopbits hinzu. Die Zeitdauer für die Stopbits bezeichnet man bei Protokoll T = 0 als Schutzzeit (*guardtime*), welche im Prinzip auch eine Art von Stopbit ist. Während dieser Zeit hat sowohl der Empfänger als auch der Sender Zeit, sich auf das nächste Byte der Übertragung vorzubereiten. Die Parität eines Bytes muss immer gerade sein. Das Paritätsbit erhält also den logischen Wert „1“, wenn die Anzahl der Einsen im Byte eine ungerade Zahl ist und umgekehrt.

Da Chipkarten-Mikrocontroller über keinen vom angelegten Takt unabhängigen Timer verfügen, kann die Dauer eines Datenbits nicht in absoluten Zeitwerten angegeben werden. Deshalb ist sie in Abhängigkeit vom angelegten Takt festgelegt. Dazu ist ein Teiler definiert,

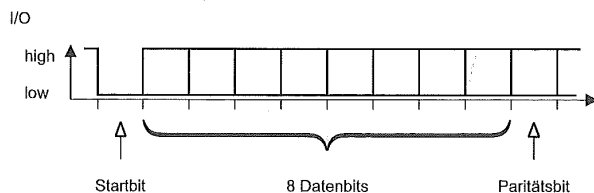


Bild 6.5 Der Aufbau eines Zeichens bei der Datentübertragung.

der die Anzahl der Takte pro Bit angibt. Die Zeitdauer für ein Bit wird als ein *etu* (*elementary time unit*) bezeichnet.

Damit ist es also bei Chipkarten unsinnig, eine Datenübertragungsrate mit einem festen Wert anzugeben (z. B. 9 600 Bit/s), da diese direkt proportional dem angelegten Takt ist. Im Wesentlichen gibt es allerdings nur zwei Teiler, die weltweit verwendet werden. Zum einen ist dies der Wert 372 und zum anderen 512. Zur Maximierung der Übertragungsgeschwindigkeit kommen seit einiger Zeit immer häufiger kleinere Teiler zur Anwendung. Bei kleinen Teilern ist es für das Betriebssystem der Karte zunehmend schwieriger, Daten zu empfangen und zu senden, da der Prozessor nur mehr sehr wenig Zeit dafür hat. So bleiben etwa beim Empfang von Daten mit einem Teiler 64 für den Prozessor nur 64 Takte Zeit, um ein Bit zu erkennen und im I/O-Puffer der Karte abzulegen.

Zur Berechnung der mit den Standard-Teilern erzielbaren Übertragungsraten muss lediglich der angelegte Takt sowie der verwendete Teiler herangezogen werden.

$$\frac{3,5712 \text{ MHz}}{372} = 9\,600 \frac{\text{Bit}}{\text{s}}$$

$$\frac{4,9152 \text{ MHz}}{512} = 9\,600 \frac{\text{Bit}}{\text{s}}$$

So erhält man beispielsweise für die beiden üblichen Standardtaktfrequenzen 3,5712 MHz und 4,9152 MHz eine Datenübertragungsrate von genau 9 600 Bit/s³. Diese 9 600 Bit/s sind auch der Grund für die zahlenmäßig krummen Teilerwerte. In der Frühzeit der Chipkartentechnik gab es nur sehr wenige Frequenzen, für die preisgünstige Quarzoszillatoren erhältlich waren. Also verwendete man billige Standard-Oszillatoren, die für Fernseher produziert wurden, und definierte die Teiler für Chipkarten so, dass man die damals schon übliche Übertragungsgeschwindigkeit von 9 600 Bit/s erhielt. Aus ähnlichen Gründen wurde bei den ersten PCs eine Taktrate von 4,77 MHz gewählt, weil diese kompatibel zum US-Farbfernsehen war und damit ein Fernseher, zumindest prinzipiell, an einen PC angeschlossen werden konnte.

Nimmt man einen maximalen Wert des Taktes von 5 MHz und einen minimalen Teiler von 32 an, so erhält man die derzeitige Obergrenze der Übertragungsrate, sofern diese durch den Prozessor in Software durchgeführt wird.

$$\frac{5 \text{ MHz}}{32} = 156\,250 \frac{\text{Bit}}{\text{s}}$$

Natürlich besteht die Möglichkeit, den Teiler noch weiter zu reduzieren, so dass die Übertragungskapazität gesteigert werden kann. Dies bedeutet jedoch einen erheblichen

³ Hier und im Folgenden wird für die Datenübertragungsrate immer die Einheit Bit/s verwendet. In der Literatur und auch in manchen Normen wird fälschlicherweise zum Teil äquivalent zu Bit/s die Einheit baud benutzt. Baud benennt während einer Datenübertragung die Anzahl von Statusänderungen pro Sekunde. Je Statusänderung können jedoch abhängig vom Übertragungsverfahren ein oder mehrere Informationsbits übertragen werden. Aus diesem Grund kann Baud nur in dem Sonderfall, dass bei jeder Statusänderung lediglich ein Bit übertragen wird, mit der Übertragungsrateneinheit bit/s gleichgesetzt werden.

Mehraufwand an Programmcode in der Chipkarte, der aus Gründen des geringen Speicherplatzes normalerweise nicht realisiert wird.

Bei vielen Neuentwicklungen von Chipkarten-Mikrocontrollern wird der Datenaustausch über die serielle Schnittstelle mit einer auf dem Chip befindlichen Hardwarebaugruppe (*universal asynchronous receiver transmitter – UART*) durchgeführt. Der Aufwand an Software für die Übertragung ist bei diesen Chiptypen nur mehr gering, und es ist möglich, die Geschwindigkeit der Datenübertragung durch die Hardwarelösung stark zu steigern. Die genormte Übertragungsrate von 111,6 kBit/s ist für eine Schnittstellenbaugruppe auf dem Chip ohne weiteres möglich.⁴

Aus den Daten für die Taktfrequenz der Karte und den Teilern lässt sich die Zeitdauer für ein Bit berechnen. Dabei erhält man bei 3,5712 MHz Takt und dem Teiler 372 eine Zeitdauer von 104 μ s, was dann per Definitionem für diesen Teiler einem etu (*elementary time unit*) entspricht. Damit kann man das folgende Zeitdiagramm zeichnen.

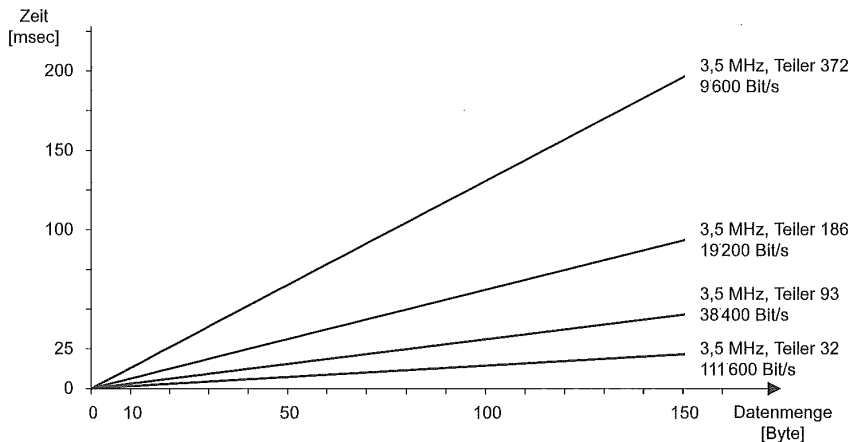


Bild 6.6 Diagramm zur Ermittlung der Übertragungszeiten von Daten bei typischen Übertragungsgeschwindigkeiten. Folgende Annahmen wurden dabei getroffen: 1 Startbit, 8 Datenbits, 1 Paritätsbit und 2 Stopbit pro Byte.

Die serielle Datenübertragung muss nicht völlig toleranzfrei ablaufen, sondern darf aus technischen Gründen gewissen Abweichungen unterliegen. Weil Chipkarten-Mikrocontroller oft noch keine Schnittstellenbausteine besitzen, muss man diese erlaubten Toleranzen in der Praxis bei den Softwarelösungen manchmal auch ausnutzen. Die Abweichung von der fallenden Flanke des Startbits bis zur abschließenden Flanke des n-ten Bits darf maximal $\pm 0,2$ etu sein. Dies bedeutet für den Sender, dass ein einzelnes Bit sehr wohl eine Abweichung von $\pm 0,2$ etu haben darf, doch auch über mehrere Bits gesehen ist eine größere Abweichung nicht erlaubt. Zeitliche Fehler bei einem Bit dürfen sich also nicht so stark aufsummieren, dass die zulässige Toleranz überschritten wird.

⁴ Siehe auch Abschnitt 16.11.3 „Tabelle zur Ermittlung der Übertragungsgeschwindigkeit“.

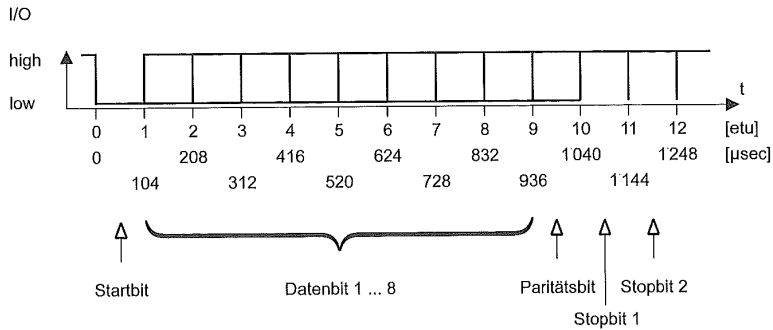


Bild 6.7 Zeitliche Darstellung eines Zeichens bei 9600 Bit/s, was beispielsweise bei einer Taktfrequenz von 3,5712 MHz und dem Teiler 372 der Fall ist.

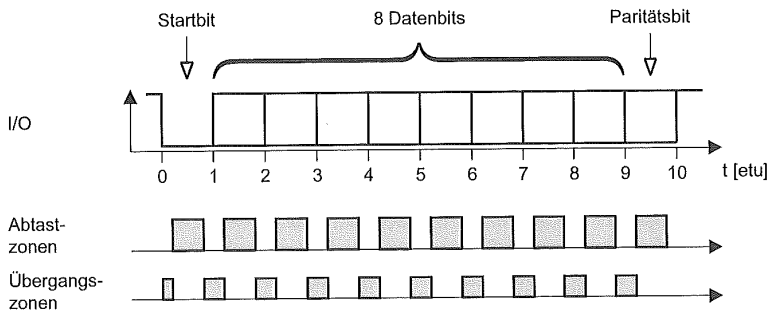


Bild 6.8 Die Abtast- und Übergangszonen beim Empfang eines Bytes.

Gerade bei der Übertragung von Daten über eine Leitung treten oft Einbrüche oder Überschwinger bei den Signalen auf. Man verlässt sich beim Empfang deshalb oft nicht nur auf eine Einfachabtastung des eingehenden Signals, sondern prüft es mehrmals. Eine typische Vorgehensweise ist eine Dreifachabfrage mit anschließender 2-aus-3-Entscheidung. Damit lassen sich kleinere Unsauberkeiten des Signalpegels bei der Übertragung ohne größeren Aufwand kompensieren. Eine weitere Steigerung auf fünf oder sieben Abtastungen würde aber bei der im Allgemeinen recht guten Datenübertragung bei Chipkarten im Verhältnis zu dem dafür nötigen Aufwand wenig Sinn ergeben.

Die drei Abfragen sollten so weit wie möglich über das zu empfangende Bit verteilt werden, damit kurzzeitige Signalschwächen optimal ausgeglichen werden können. Damit stößt man links und rechts vom Bitmittelpunkt auf die Grenzen, die durch die maximal erlaubten Toleranzen beim Senden eines Bytes gesetzt sind (Abtastzone – *test zone*). Durch Ausmitteln der unteren und oberen Grenzen und des Bitmittelpunktes erhält man die optimalen Abtastzeitpunkte bei einer Dreifachabtastung. Dies unterliegt aber keiner Normung.⁵ In den Übergangszonen (*transition zone*) darf nicht abgetastet werden, da das dort auftretende Signal ungültig ist.

⁵ Siehe auch Abschnitt 16.11.4 „Tabelle mit Abtastzeitpunkten“.

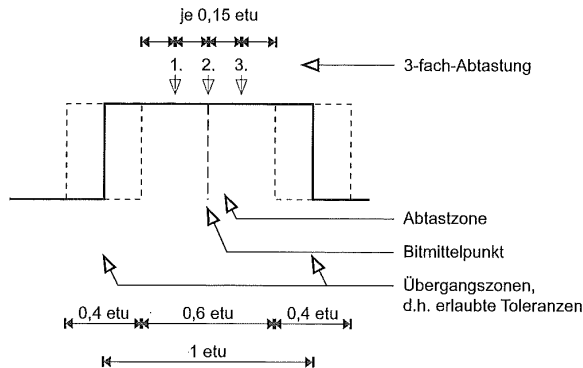


Bild 6.9 Beispiel für eine 3-fach-Abtastung eines Bits beim Empfang.

6.2 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 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.

Es kommt sehr selten vor, dass ein ATR die maximale Länge erreicht. Viel öfter ist der ATR nur aus sehr 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 ist die Bezahlung von Maut an Autobahnen mit elektronischen Gelddörren in Form von Chipkarten. Trotz hoher Geschwindigkeit der Fahrzeuge beim Durchfahren der Erfassungsstellen muss in der kurzen zur Verfügung stehenden Zeit ein sicheres Abbuchen möglich sein.

Mit dem Aussenden des ATR muss zwischen 400 und 40000 Taktzyklen nach dem Freigeben des Resetsignals durch das Terminal begonnen werden. Dies entspricht bei einer Taktfrequenz von 3,5712 MHz der Zeitspanne von 112 μ s bis 11,20 ms, und bei 4,9152 MHz sind es 81,38 μ s bis 8,14 ms.⁶ Empfängt das Terminal während dieses Zeitraums nicht den Beginn des ATRs, so wiederholt es mehrmals (üblicherweise dreimal) die Einschaltsequenz 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 9600 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. Das bedeutet, dass es nach der Norm erlaubt ist, zwischen den einzelnen Bytes im ATR

⁶ Siehe auch Abschnitt 16.11.2 „Umrechnungstabelle für Datenelemente des ATR“.

jeweils eine ganze Sekunde zu warten und erst dann das nächste Byte zum Terminal zu senden. In manchen 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.⁷

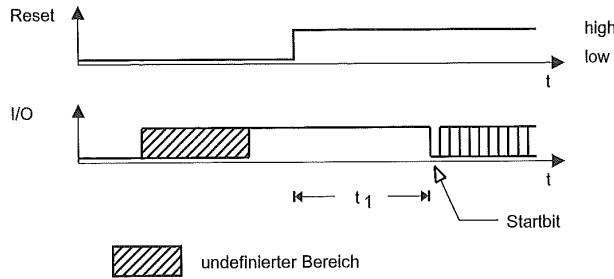


Bild 6.10 Zeitlicher Ablauf des Resetsignals und Start des ATRs nach ISO/IEC 7816-3 mit 400 Takte $\leq t_1 \leq 40000$ Takte.

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 spezielle Übertragungsparameter 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.

Tabelle 6.1 Die Datenelemente des ATR und ihre Bezeichnungen nach ISO/IEC 7816-3

Datenelement	Bezeichnung
TS	The Initial Character
T0	The Format Character
TA1, TB1, TC1, TD1, ...	The Interface Characters
T1, T2, ..., TK	The Historical Characters
TCK	The Check Character

⁷ Siehe auch Abschnitt 5.10 „Atomare Abläufe“.

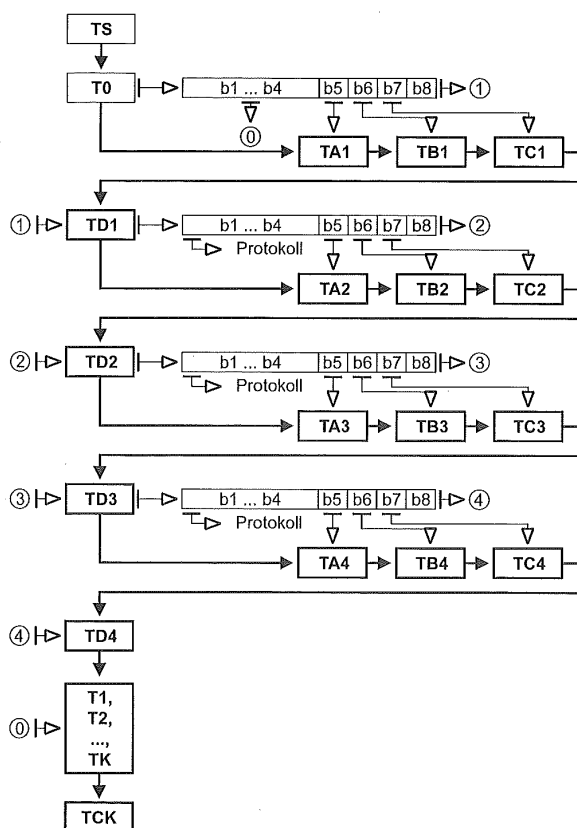


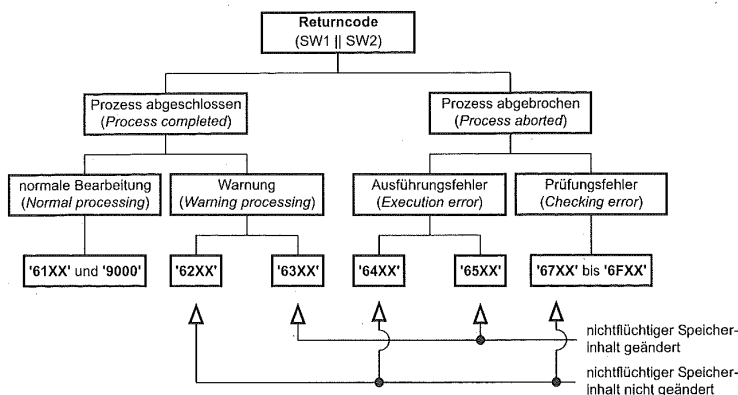
Bild 6.11 Der grundlegende Aufbau und die Datenelemente des ATR.

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 Codierungen: '3B' in direct convention oder '3F' in inverse convention codiert.

Die in Deutschland übliche convention der Bytes ist direct. In Frankreich wird vornehmlich inverse convention benutzt. Die convention hat keinen Einfluss auf die Sicherheit 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.

Systematik der Returncodes



Wichtige Returncodes

Returncode	Bedeutung	Returncode	Bedeutung
'61xx'	Kommando erfolgreich ausgeführt, xx Byte Daten sind als Antwort vorhanden und können mit GET RESPONSE angefordert werden.	'6A00'	Falsche Parameter P1/P2 (allgemein)
'6281'	Die zurückgegebenen Daten können u.U. fehlerhaft sein	'6A80'	Parameter im Datenteil sind falsch
'6282'	Es konnten weniger als Le Bytes gelesen werden, da das Datelende vorher erreicht wurde	'6A81'	Funktion wird nicht unterstützt
'6283'	Die selektierte Datei ist reversibel gesperrt (invalidated).	'6A82'	Datei wurde nicht gefunden
'6284'	Die File Control Information (FCI) ist nicht nach ISO/IEC 7816-4 strukturiert.	'6A83'	Record wurde nicht gefunden
'62xx'	Warnung; Zustand des nichtflüchtigen Speichers unverändert.	'6A84'	Ungenügend Speicherplatz in der Datei
'63Cx'	Zähler hat den Wert x erreicht ($0 \leq x \leq 15$, (die genaue Bedeutung ist vom jeweiligen Kommando abhängig)	'6A85'	Lc inkonsistent mit TLV Struktur
'63xx'	Warnung; Zustand des nichtflüchtigen Speichers verändert	'6A86'	Inkorrekte Parameter P1/P2
'64xx'	Ausführungsfehler; Zustand des nichtflüchtigen Speichers unverändert	'6A87'	Lc inkonsistent mit P1/P2
'6581'	Speicherfehler (z.B. bei Schreiboperation)	'6A88'	referenzierte Daten nicht gefunden
'65xx'	Ausführungsfehler; Zustand des nichtflüchtigen Speichers verändert	'6B00'	Parameter 1 oder 2 falsch
'6700'	Länge falsch	'6Cxx'	falsche Länge Le; xx gibt die korrekte Länge an
'67xx' ... '6Fxx'	Prüfungsfehler	'6D00'	Kommando (Instruction) wird nicht unterstützt
'6800'	Funktionen im Class Byte werden nicht unterstützt (allgemein)	'6E00'	Class wird nicht unterstützt
'6881'	Logische Kanäle werden nicht unterstützt	'6F00'	Kommando abgebrochen - Genauere Diagnose nicht möglich (z.B. Fehler im Betriebssystem)
'6882'	Secure Messaging wird nicht unterstützt	'9000'	Kommando erfolgreich ausgeführt
'6900'	Kommando nicht erlaubt (allgemein)	'920x'	Schreiben ins EEPROM nach x-maligem Versuch erfolgreich
'6981'	Kommando inkompatibel zur Dateistruktur	'9210'	Ungenügend Speicherplatz
'6982'	Sicherheitszustand nicht erfüllt	'9240'	Schreiben ins EEPROM nicht erfolgreich
'6983'	Authentisierungsmethode gesperrt	'9400'	kein EF selektiert
'6984'	referenzierte Daten sind reversibel gesperrt (invalidated)	'9402'	Adressbereich überschritten
'6985'	Benutzungsbedingungen nicht erfüllt	'9404'	FID nicht gefunden, Record nicht gefunden, Vergleichsmuster nicht gefunden
'6986'	Kommando nicht erlaubt (kein EF selektiert)	'9408'	Selektierter Dateityp unpassend zum Kommando
'6987'	erwartete Secure Messaging Datenobjekte fehlen	'9802'	Keine PIN definiert
'6988'	Secure Messaging Datenobjekte inkorrekt	'9804'	Zugriffsbedingungen nicht erfüllt, Authentisierung fehlgeschlagen
		'9835'	ASK RANDOM / GIVE RANDOM nicht ausgeführt
		'9840'	PIN-Prüfung nicht erfolgreich
		'9850'	INCREASE / DECREASE kann nicht ausgeführt werden, da Grenzwert erreicht
		'9Fxx'	Kommando erfolgreich ausgeführt, xx Byte Daten sind als Antwort vorhanden und können mit GET RESPONSE angefordert werden.

Kodierung wichtiger Kommandos nach ISO/IEC 7816-4

SELECT FILE (case 4)

Datenelement	Codierung	Bemerkung
CLA	'00'	Das für ISO/IEC 7816 Kommandos reservierte Class-Byte ohne Benutzung von Secure Messaging.
INS	'A4'	Das Instruction-Byte für SELECT FILE, d.h. das Kommando zur Auswahl von Dateien (MF, DFs oder EFs).
P1	...	$P1 = '00' \wedge Lc = 0$ Selektion des MF $P1 = '00' \wedge Lc \neq 0$ Selektion einer Datei mit FID (FID in DATA enthalten) $P1 = '04'$ Selektion eines DF mit dem DF Name (DF Name in DATA enthalten) $P1 = '08'$ Selektion einer Datei durch die Angabe des FID basierten Pfades (Pfad in DATA enthalten) vom MF aus. $P1 = '09'$ Selektion einer Datei durch die Angabe des FID basierten Pfades (Pfad in DATA enthalten) vom aktuellen DF aus.
P2	...	$P2 = '00'$ Rückgabe von optionalen FCI. $P2 = '04'$ Rückgabe von optionalen FCP. $P2 = '08'$ Rückgabe von optionalen FMD.
Lc	...	Codierung in der Beschreibung von P1 enthalten.
DATA	...	Codierung in der Beschreibung von P1 enthalten.
Le	...	$Le = 0'$ Rückgabe aller zur Selektion gehörenden Daten.

READ BINARY (case 2)

Datenelement	Codierung	Bemerkung
CLA	'00'	Das für ISO/IEC 7816 Kommandos reservierte Class-Byte ohne Benutzung von Secure Messaging.
INS	'B0'	Das Instruction-Byte für READ BINARY, d.h. das Kommando zum Lesen von Daten aus Dateien der Struktur „transparent“.
P1	...	$P1.b8 = 0$ Lese Daten aus der aktuell selektierten Datei mit Offset; Offset = (P1.b7 ... P1.b1 P2) $P1.b8 = 1$ Lese Daten nach impliziter Dateiselektion durch Short-FID und mit Offset; Short-FID = (P1.b5 ... P1.b1), Offset = P2
P2	...	Codierung in der Beschreibung von P1 enthalten.
Le	...	$Le = 0$ Lese alle Daten bis zum Ende der Datei. $Le > 0$ Le ist die Anzahl der zu lesenden Bytes.

UPDATE BINARY (case 3)

Datenelement	Codierung	Bemerkung
CLA	'00'	Das für ISO/IEC 7816 Kommandos reservierte Class-Byte ohne Benutzung von Secure Messaging.
INS	'D6'	Das Instruction-Byte für UPDATE BINARY, d.h. das Kommando zum Schreiben von Daten in Dateien der Struktur „transparent“.
P1	...	$P1.b8 = 0$ Schreibe Daten in die aktuell selektierte Datei mit Offset; Offset = (P1.b7 ... P1.b1 P2) $P1.b8 = 1$ Schreibe Daten nach impliziter Dateiselektion durch Short-FID und mit Offset; Short-FID = (P1.b5 ... P1.b1), Offset = P2
P2	...	Codierung in der Beschreibung von P1 enthalten.
Lc	...	Lc ist die Anzahl der zu schreibenden Bytes.
DATA	...	Die zu schreibenden Bytes mit der Länge Lc.

Kodierung wichtiger Kommandos nach ISO/IEC 7816-4

READ RECORD (case 2)

Datenelement	Codierung	Bemerkung
CLA	'00'	Das für ISO/IEC 7816 Kommandos reservierte Class-Byte ohne Benutzung von Secure Messaging.
INS	'B2'	Das Instruction-Byte für READ RECORD, d.h. das Kommando zum Lesen von Daten aus einer Datei mit Record-orientierter Struktur.
P1	...	P1 = 0 Lese den aktuellen Record P1 ≠ 0 Lese den Record mit der in P1 angegebenen Recordnummer oder Record Identifier.
P2	...	P2.b8 ... P2.b4 = "00000" Lese Daten aus der aktuell selektierten Datei P2.b8 ... P2.b4 ≠ "00000" Lese Daten nach impliziter Dateiselektion durch Short-FID; Short-FID = (P2.b8 ... P2.b4) P2.b3 ... P2.b1 = "000" Lese den ersten (<i>first</i>) Record mit dem Record Identifier, übergeben in P1 P2.b3 ... P2.b1 = "001" Lese den letzten (<i>last</i>) Record mit dem Record Identifier, übergeben in P1 P2.b3 ... P2.b1 = "010" Lese den nächsten (<i>next</i>) Record mit dem Record Identifier, übergeben in P1 P2.b3 ... P2.b1 = "011" Lese den vorherigen (<i>previous</i>) Record mit dem Record Identifier, übergeben in P1 P2.b3 ... P2.b1 = "100" ∧ P1 = 0 Lese den aktuellen Record P2.b3 ... P2.b1 = "100" ∧ P1 ≠ 0 Lese den Record mit der Recordnummer, übergeben in P1. P2.b3 ... P2.b1 = "101" Lese alle Records ab der in P1 übergebenen Recordnummer bis zum Ende der Datei. P2.b3 ... P2.b1 = "110" Lese alle Records vom Ende der Datei bis zu der in P1 übergebenen Recordnummer.
Le	...	Le = 0 Lese alle Bytes bis zum Ende des Records/der Records. Le > 0 Le ist die Länge des Records/der Records.

UPDATE RECORD (case 3)

Datenelement	Codierung	Bemerkung
CLA	'00'	Das für ISO/IEC 7816 Kommandos reservierte Class-Byte ohne Benutzung von Secure Messaging.
INS	'DC'	Das Instruction-Byte für UPDATE RECORD, d.h. das Kommando zum Schreiben von Daten in eine Datei mit Record-orientierter Struktur.
P1	...	P1 = 0 Schreibe den aktuellen Record P1 ≠ 0 Schreibe den Record mit der in P1 übergebenen Recordnummer.
P2	...	P2.b8 ... P2.b4 = "00000" Schreibe Daten in die aktuell selektierte Datei P2.b8 ... P2.b4 ≠ "00000" Schreibe Daten nach impliziter Dateiselektion durch Short-FID; Short-FID = (P1.b8 ... P1.b4) P2.b3 ... P2.b1 = "000" Schreibe den ersten (<i>first</i>) Record P2.b3 ... P2.b1 = "001" Schreibe den letzten (<i>last</i>) Record P2.b3 ... P2.b1 = "010" Schreibe den nächsten (<i>next</i>) Record P2.b3 ... P2.b1 = "011" Schreibe den vorherigen (<i>previous</i>) Record P2.b3 ... P2.b1 = "100" Schreibe den Record mit der in P1 übergebenen Recordnummer
Lc	...	Lc ist die Länge des zu schreibenden Records.
DATA	...	Der zu schreibende Record.

VERIFY (case 3)

Datenelement	Codierung	Bemerkung
CLA	'00'	Das für ISO/IEC 7816 Kommandos reservierte Class-Byte ohne Benutzung von Secure Messaging.
INS	'20'	Das Instruction-Byte für VERIFY, d.h. das Kommando zum Vergleich von übergebenen Daten mit Referenzdaten (typischerweise einer PIN).
P1	'00'	---
P2	...	P2 = '00' Explizite Informationen werden nicht übergeben. P2.b8 = "0" Es werden für die ganze Chipkarte gültige Referenzdaten verwendet (<i>global reference data</i>). P2.b8 = "1" Es werden für eine oder mehrere Anwendungen gültige Referenzdaten verwendet (<i>specific reference data</i>). P2.b7 P2.b6 = "00" RFU Bits P2.b5 ... P2.b1 Nummer der Referenzdaten.
Lc	...	Lc ist die Länge des übergebenen Vergleichswertes.
DATA	...	Der übergebene Vergleichswert (in der Regel eine PIN).

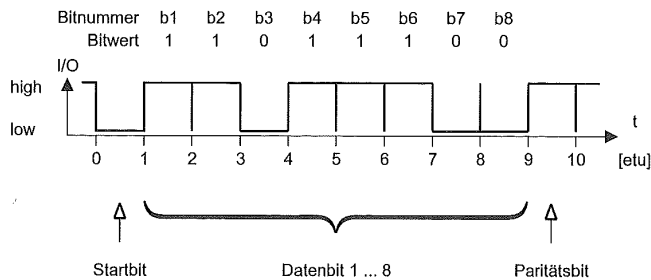


Bild 6.12 Zeitliche Darstellung des Initial Character TS bei direct convention. Dies wird durch den Wert '3B' = °0011 1011° angezeigt.

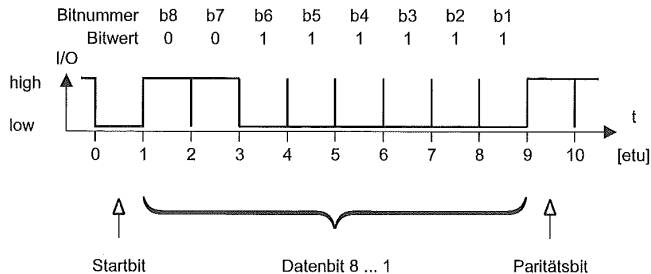


Bild 6.13 Zeitliche Darstellung des Initial Character TS bei inverse convention. Dies wird durch den Wert '3F' = °0011 1111° angezeigt.

Tabelle 6.2 Die Codierung des Initial Characters TS

b8	b7	b6	b5	b4	b3	b2	b1	Bedeutung
								direct convention
								inverse convention

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 6.3 Die Codierung des Format Characters T0

b8	b7	b6	b5	b4	b3	b2	b1	Bedeutung
...	...	...	...	X	X	X	X	Anzahl der Historical Characters [0 ... 15]
...	...	...	1	...	...	...	...	TA1 wird übertragen
...	...	1	...	...	...	...	...	TB1 wird übertragen
...	1	...	...	...	...	...	...	TC1 wird übertragen
1	...	...	...	...	...	...	...	TD1 wird übertragen

Die Interface Characters

Die Interface Characters spezifizieren alle Übertragungsparameter der vorhandenen Übertragungsprotokolle. Sie setzen sich aus den Bytes TAI, TBI, TCI und TDI zusammen. Die Übertragung dieser Bytes im ATR ist jedoch optional und kann damit gegebenenfalls auch unterbleiben. Da für alle Übertragungsparameter der Übertragungsprotokolle grundlegende Werte definiert sind, benötigt man deshalb 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 Interface Characters nur für T = 0 relevant. Falls T = 0 nicht realisiert ist, kann man sie weglassen. Dann gelten die voreingestellten Werte.

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

Ist das Byte TDI nicht vorhanden, dann bedeutet dies, dass TAI + 1, TBI + 1, TCI + 1 und TDI + 1 nicht übertragen werden.

Tabelle 6.4 Die Codierung eines TDI-Bytes

b8	b7	b6	b5	b4	b3	b2	b1	Bedeutung
...	...	...	...	X				Nummer des Übertragungsprotokolls [0 . . . 15]
...	...	...	1	...				TAi + 1 wird übertragen
...	...	1	...	...				TBi + 1 wird übertragen
...	1	...	...	...				TCi + 1 wird übertragen
1	...	...	...	...				TDi + 1 wird übertragen

Die nicht der Verknüpfung dienenden anderen Bytes der Interface Characters TAI, TBI und TCI spezifizieren das oder die vorhandenen Übertragungsprotokolle. Sie haben nach ISO/IEC 7816-3 die folgenden Bedeutungen:

Global Interface Character TAI

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

Tabelle 6.5 Die Codierung von TA1

b8	b7	b6	b5	b4	b3	b2	b1	IFSC
		X				...		FI
		...				X		DI

Tabelle 6.6 Die Codierung von FI

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

Tabelle 6.7 Die Codierung von DI

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

Aufgrund obiger Codierungen des Teilers F und des Übertragungsanpassungsfaktors D können die typischen Übertragungsgeschwindigkeiten normgerecht festgelegt werden. Im Abschnitt 16.11.2 (Umrechnungstabelle für Datenelemente des ATR) befindet sich eine Tabelle zur Übersicht.

Es gelten dabei die nachfolgenden Zusammenhänge. Die Zeitdauer eines Bits beim ATR und PPS, als „initial etu“ bezeichnet, ist auf folgenden Wert festgesetzt:

$$\text{initial etu} = \frac{372}{f_i} \cdot s$$

Im 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

$$\text{work etu} = \frac{1}{D} \cdot \frac{F}{f_s} \cdot s$$

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 *f_{max}* gibt die maximal erlaubte Taktfrequenz an. Der Standardwert für *f_{max}* ist 5 MHz.

Global Interface Character TA(i)

Der Wert TA(i) (= XI || UI) ist g¼ltig f¼r immer $i > 2$ und $T = 15 = 'F'$ in TD(i-1). TA(i) enth¼lt den Indikator XI zum Anhalten des Takts (*clock stopp indicator*). Dieser gibt an, auf welchem logischen Zustand sich die Taktleitung befinden muss, wenn der Takt angehalten wird. Zus¼tzlich enth¼lt TA(i) einen Indikator (*class indicator*) f¼r die Klasse der Versorgungsspannung.

Tabelle 6.8 Die Codierung von XI

X	nicht unterst¼tzt	Zustand low	Zustand high	kein bevorzugter Zustand
XI	00	01	10	11

Tabelle 6.9 Die Codierung von UI

U	Spannungsklasse A 4,5 V–5,5 V	Spannungsklasse B 2,7 V–3,3 V	Spannungsklassen A und B	RFU
UI	00 0001	00 0010	00 0011	alle anderen Werte

Global Interface Character TB1

In diesem Byte ist in Bit b7 und b6 ein Faktor II f¼r die Programmierspannung codiert. Die Bits b5 bis b1 definieren PI1. Das h¼herwertigste Bit b8 ist immer auf 0 gesetzt, d. h. es wird nicht verwendet.

Tabelle 6.10 Die Codierung von TB1

b8	b7	b6	b5	b4	b3	b2	b1	IFSC
0		X	...	...	...	...	...	II
0		...	X	X	X	X	X	PI1

Diese Parameter wurden in der ersten Generation von Chipkarten ben¼tigt, die statt des heute ¼blichen EEPROM ein EPROM als Datenspeicher hatten. Die bei der Programmierung des EPROM erforderlichen hohen Spannungen und Str¼me mussten vom Terminal ¼ber den Vpp-Kontakt zugef¼hrt werden. Da jedoch Chipkarten ohne interne Ladungspumpe nicht mehr existieren, wurde hier auf die ausf¼hrliche Codierung verzichtet.

Die Datenelemente PI1 und II haben dann immer den Wert 0, was signalisiert, dass keine externe Programmierspannung erforderlich ist. Wird auf das Datenelement TB1 im ATR verzichtet, so gilt die in der Norm festgelegte Grundeinstellung von 5 Volt und 50 mA f¼r die Vpp Versorgung.

Global Interface Character TC1

Hier wird eine zus¼tzliche Schutzzeit (*extra guardtime*) mit der Bezeichnung N als hexadezimaler vorzeichenloser und ganzzahliger Bytewert codiert. Die zus¼tzliche Schutzzeit ist dabei definiert als eine zeitliche Verl¼ngerung der Stoppbits. Der Wert N gibt die Anzahl der etu an, um die die Schutzzeit verl¼ngert werden soll. TC1 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 zirka 10 %, da ein Bitweniger gesendet werden muss.

Tabelle 6.11 Die Codierung von TC1

b8	b7	b6	b5	b4	b3	b2	b1	IFSC
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

Global Interface Character TB2

In diesem Byte befindet sich der Wert von PI2. Das Datenelement gibt die externe Programmierspannung in zehntel Volt an. Üblicherweise wird es im ATR aus demselben Grund wie TB1 nicht mehr verwendet.

Tabelle 6.12 Die Codierung von TB2

b8	b7	b6	b5	b4	b3	b2	b1	IFSC
X								PI2

Für das Übertragungsprotokoll T = 0 ist noch der folgende Specific Interface Character definiert:

Specific Interface Character TC2

Das letzte Datenelement des T = 0 Protokolls codiert die work waiting time und wird als TC2 bezeichnet. Die work waiting time ist die maximale Zeit zwischen den Startflanken zweier aufeinander folgender Bytes.

$$\text{work waiting time} = (960 \cdot D \cdot WI) \text{ work etu}$$

Falls TC2 im ATR nicht enthalten ist, beträgt der Standardwert für die work waiting time WI = 10.

Tabelle 6.13 Die Codierung von TC2

b8	b7	b6	b5	b4	b3	b2	b1	Bedeutung
X								WI

Für das Übertragungsprotokoll T = 1 sind die folgenden zusätzlichen Bytes entsprechend ISO/IEC 7816-3 definiert. Die für T = 0 vorgeschriebenen Interface Characters finden hier

nur bei Bedarf Verwendung. Für dieses Protokoll muss der Index i bei den Datenelementen immer größer als zwei sein. Die Specific Interface Characters TAI, TBI, TCi ($i > 2$) gelten dabei immer für dasjenige Übertragungsprotokoll, das in $TD_{(i-1)}$ angegeben ist.

Specific Interface Character TAI ($i > 2$)

Das Byte TAI 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 6.14 Die Codierung von TAI für $i > 2$

b8	b7	b6	b5	b4	b3	b2	b1	Bedeutung
				X				IFSC

Specific Interface Character TBI ($i > 2$)

Das niederwertige Halbbyte beinhaltet in den vier Bits b4 bis b1 die Codierung für die Zeichenwartezeit CWT⁸, sodass gilt:

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

Das höherwertige Halbbyte enthält den Wert BWI, mit dem die Blockwartezeit BWT⁹ folgendermaßen errechnet werden kann:

$$BWT = 2^{BWI} \cdot 960 \cdot \frac{372}{f} \cdot s + 11 \text{ work etu}$$

Tabelle 6.15 Die Codierung von TBI für $i > 2$

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

Specific Interface Character TCi ($i > 2$)

Das Bit b1 codiert die Berechnungsweise des Fehlererkennungscode. Da in der Norm zu den Datenelementen des ATRs nicht alle möglichen Parameter der Übertragungsprotokolle im Rahmen der Interface Characters definiert sind, können verschiedene Implementierungen zusätzliche Interface Characters benutzen.

Ein typisches Beispiel dafür ist das nationale Protokoll T = 14 in Deutschland. Dort sind noch einige zusätzliche Bytes im ATR für die spezifischen Belange dieses Protokolls definiert. Die Decodierung ist jedoch dann nur für die Anwender dieses Protokolls möglich, da sie nur unter Kenntnis der entsprechenden Spezifikation erfolgen kann. Dies ist dann aber weder genormt noch außerhalb der jeweiligen Anwendung bekannt.

⁸ Siehe auch Abschnitt 6.4.3 „Übertragungsprotokoll T = 1“.

⁹ Siehe auch Abschnitt 6.4.3 „Übertragungsprotokoll T = 1“.

Tabelle 6.16 Die Codierung von TCi 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

Global Interface Character TA2

Dieses Byte zeigt die möglichen Modi an, die für den PPS verwendet werden können. Im Abschnitt 6.4 über den PPS ist dies genauer erläutert.
Die Codierung von TA2 ist wie folgt:

Tabelle 6.17 Die Codierung von TA2

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

Die Historical Characters

Der Inhalt der Historical Characters war lange Zeit von keiner Norm festgelegt, was dazu führte, dass sie je nach Betriebssystemhersteller die verschiedensten Daten enthalten.

Viele Firmen geben in den zur Verfügung stehenden Bytes eine Bezeichnung für das Betriebssystem sowie eine dazugehörige Versionsnummer der ROM-Maske an. Die Codierung ist meist in ASCII, sodass 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' enthält weitere Informationen über den ATR. Sie ist als eine 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-codiert.

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 Codierung der entsprechenden Datenobjekte ist in den Normen ISO/IEC 7816-4 und -5 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 6.18 Die Codierung des Category Indicators in T1

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

Tabelle 6.19 Die Codierung der anwendungsunabhängigen Kartendienste im Compact-TLV codierten 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-/ATR-Datei mittels Kommando READ RECORD lesbar.
...	...	...	...	...	0	0	0	nicht benutzt

Tabelle 6.20 Die Codierung der Kartenfähigkeiten (*first software function table*) in Compact-TLV codierten Datenobjekt(e). 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-Identifizier werden unterstützt.

Tabelle 6.21 Die Codierung der Kartenfähigkeiten (*second software function table*) in Compact-TLV codierten Datenobjekt(e). 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 (<i>one time write</i>).
...	0	1	...	...	...	...	...	Das WRITE Kommando verhält sich anwendungsspezifisch (<i>proprietary</i>).
...	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 ($^{\circ}001^{\circ} = 2^2 = 2 \text{ Nibbles} = 1 \text{ Byte}$)
0	...	...	0	0	...	...	...	nicht benutzt

Tabelle 6.22 Die Codierung der Kartenfähigkeiten (*third software function table*) in Compact-TLV codierten Datenobjekt(e). 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	...	...	...	...	...	...	Erweiterte Lc-/Le-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	Maximale Zahl der logischen Kanäle ($= 2 \cdot x + y + 1$).
0	...	0	...	...	0	...	...	nicht benutzt

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 die Prüfsumme TCK nicht am Ende des ATRs befinden. Sie wird dann überhaupt 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.

Praxisbeispiele für ATRs

Die folgenden Bilder 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 6.23 Beispiel für einen ATR einer Chipkarte mit Übertragungsprotokoll T = 1

Bezeichnung	Wert	Bedeutung	Bemerkung
TS	'3B'	direct convention	
T0	'B5'	Y1 = 1011 = 'B' K = '5'	TA1, TB1, TD1 folgen 5 Historical Characters
TA1	'11'	FI = 0001 = '1' DI = 0001 = '1'	F = 372 D = 1

Tabelle 6.23 (Fortsetzung)

TB1	'00'	II = 0 PI1 = 0000 = '0'	I = 0 Vpp-PIN ist nicht belegt
TD1	'81'	Y2 = 1000 = '8' T = 1	TD2 folgt
TD2	'31'	Y2 = 0011 = '3' T = 1	TA3 und TB3 folgen
TA3	'46'	I/O Pufferlänge = 70 Byte	Länge des ICC I/O Puffers (Schicht 7)
TB3	'15'	BWI = '1' CWI = '5'	BWT = 2 011 etu CWT = 43 work etu
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 6.24 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 = 'E' = °1001° K = 'C' = 12	TA1 und TD1 folgen 12 Historical Characters
TA1	'11'	FI = 0001 = '1' DI = 0001 = '1'	F = 372 D = 1
TD1	'81'	'2' = °1000° T = 1	TD2 folgt T = 1 wird verwendet
TD2	'21'	'2' = °0010° T = 1	TB2 folgt T = 1 wird verwendet
TB2	'34'	CWI = 3 BWI = 4	Character Waiting Time Block 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 6.25 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° K = 'B' = 11	TA1 und TD1 folgen 11 Historical Characters
TA1	'11'	FI = 0001 = '1' DI = 0001 = '1'	F = 372 D = 1
TD1	'81'	'2' = °1000° T = 1	TD2 folgt T = 1 wird verwendet

Tabelle 6.25 (Fortsetzung)

TD2	'21'	'2'	= °0010°	TB2 folgt
		T	= 1	T = 1 wird verwendet
TB2	'34'	CWI	= 3	Character Waiting Time
		BWI	= 4	Block Waiting Time
T1 ... T15	'53' '54' '41' '52' '43' '4F' '53' '32' '31' '20' '43'			"STARCOS 21 C"
TCK	'43'	The Check Character		XOR-Prüfsumme von T0 bis einschließlich T15

Tabelle 6.26 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° K = 9	TD1 folgt 9 Historical Characters
TD1	'40'	'4' = °0100° T = 0	TC2 folgt T = 0 wird verwendet.
TC2	'14'	WI = '14'	Die Work Waiting Time ist '14'.
T1 ... T9	'47' '47' '32' '34' '4D' '35' '32' '38' '30'		"GG24M5280"

Tabelle 6.27 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	TB1 folgt 15 Historical Characters
TB1	'00'	PII = °00000° II = °00°	Vpp ist nicht benutzt, die Programmiervspannung 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 6.28 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	TB1, TC1 und TD1 folgen 3 Historical Characters
TB1	'00'	PII = °00000° II = °00°	Vpp ist nicht benutzt, die Programmiervspannung für das EEPROM wird im Chip erzeugt.
TC1	'00'	N = 0	keine extra Guardtime
TD1	'81'	'8' = °1000° T = 1	TD2 folgt. T = 1 wird verwendet.

Tabelle 6.28 (Fortsetzung)

TD2	'31'	'3' T	= °0011° = 1	TA3 und TB3 folgen. T = 1 wird verwendet.	
TA3	'6F'	IFSC	= '6F' = 111	Information Field Size der Chipkarte ist 111 Byte	
TB3	'45'	CWI BWI = 5	= 4	Character Waiting Time Block 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 Daten- objekts für die Card-Service-Daten	
T3	'C0'	'C0'	= °1100 0000°		Anwendungsselektion mit DF Name als Gan- zem oder in Teilen
TCK	'08'	The Check Character		XOR-Prüfsumme von T0 bis einschließlich T3	

6.3 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. Der Protocol Parameter Selection (PPS) wurde vor 1997 Protocol Type Selection genannt.

Der 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 diese im TA2 Byte an.

Die PPS-Anfrage (PPS-Request) muss unmittelbar nach Empfang des ATR vom Terminal durchgeführt werden. 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.

Tabelle 6.29 Die Datenelemente des PPS und ihre Bezeichnungen nach ISO/IEC 7816-3

Datenelement	Bezeichnung
PPSS	The Initial Character
PPS0	The Format Character
PPS1, PPS2, PPS3	The Format Characters
PCK	The Check Character

Die Datenelemente PPS2 und PPS3 sind für zukünftige Anwendungen reserviert und können deshalb hier noch nicht beschrieben werden.

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

Tabelle 6.30 Die Codierung von PPS0

b8	b7	b6	b5	b4	b3	b2	b1	Bedeutung
...	...	...	...			X		zu verwendendes Übertragungsprotokoll
...	...	...	1			...		PPS1 ist vorhanden
...	...	1	...			...		PPS2 ist vorhanden
...	1	...	...			...		PPS3 ist vorhanden
0	...	...	...			...		Reserviert für zukünftige Anwendung

Tabelle 6.31 Die Codierung von PPS1

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

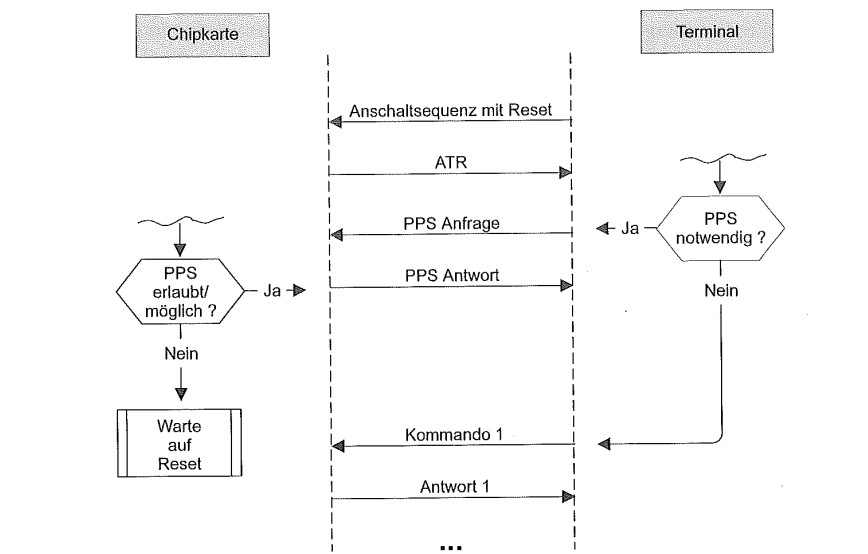


Bild 6.16 Ein typischer PPS-Ablauf einer GSM-Karte.

Kann die Karte den PPS interpretieren und das Übertragungsprotokoll 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 große Zeitverlust bis zum Einsatz des eigentlichen Übertragungsprotokolls.

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 Weitschalten des Übertragungsprotokolls.

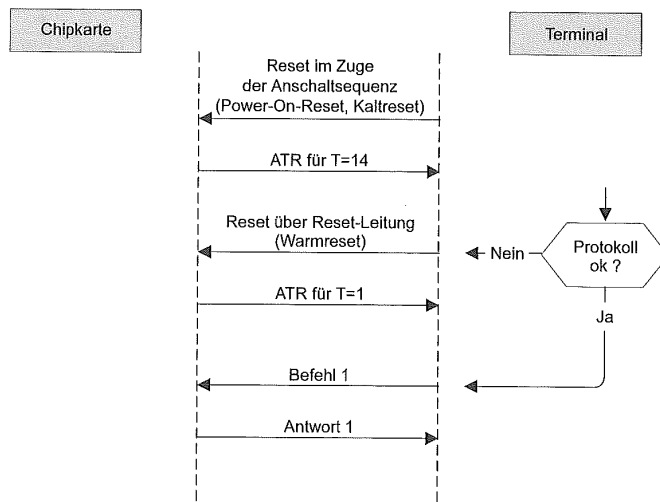


Bild 6.17 Ein typischer PPS-Ablauf über Reset.

6.4 Übertragungsprotokolle

Nachdem die Chipkarte einen ATR ausgesandt und eventuell ein PPS stattgefunden hat, wartet sie auf das erste Kommando des Terminals. Der nun folgende Ablauf entspricht immer dem Master-Slave-Prinzip mit dem Terminal als Master und der Karte als Slave. Konkret sendet das Terminal ein Kommando zur Karte, diese führt es aus und sendet anschließend die Antwort zurück. Dieses Ping-Pong-Spiel von Kommando und Antwort wird niemals durchbrochen.

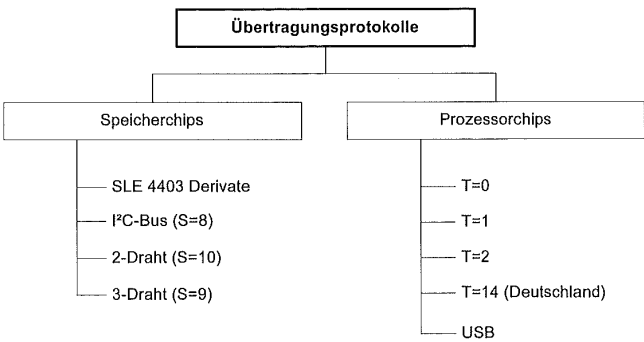


Bild 6.18 Die Systematik der kontaktbehafteten Übertragungsprotokolle von Chipkarten.

Nun gibt es aber verschiedene Möglichkeiten, die Kommunikation mit einer Chipkarte aufzubauen. Auch im Falle einer Übertragungsstörung existieren unterschiedliche Verfahren, um beide Kommunikationspartner wieder zu synchronisieren. Die genaue Realisierung der Kommandos, der dazugehörigen Antworten und das Verhalten bei Übertragungsfehlern findet in den Beschreibungen der so genannten Übertragungsprotokolle seinen Niederschlag.

Es sind insgesamt 15 Übertragungsprotokolle vorgesehen und in ihrer Grundfunktionalität definiert. Die Bezeichnung setzt sich aus dem Ausdruck „T =“ (*transport protocol*) und einer nachfolgenden Nummer zusammen. Die folgende Tabelle zeigt einen Überblick.

Tabelle 6.32 Die Übertragungsprotokolle nach ISO/IEC 7816-3 im Überblick

Übertragungsprotokoll	Bedeutung
T = 0	asynchron, halbduplex, byteorientiert, spezifiziert in ISO/IEC 7816-3
T = 1	asynchron, halbduplex, blockorientiert, spezifiziert in ISO/IEC 7816-3 Amd. 1
T = 2	asynchron, vollduplex, blockorientiert, spezifiziert in ISO/IEC 10 536-4
T = 3	vollduplex, noch nicht spezifiziert
T = 4	asynchron, halbduplex, byteorientiert, Erweiterung von T = 0, noch nicht spezifiziert
T = 5 . . . T = 13	reserviert für zukünftige Anwendungen und noch nicht spezifiziert
T = 14	für nationale Anwendung, nicht von ISO genormt
T = 15	reserviert für zukünftige Erweiterungen und noch nicht spezifiziert

Dabei haben sich international zwei Übertragungsprotokolle durchgesetzt. Zum einen ist dies das $T = 0$ Protokoll, das im Jahr 1989 international genormt wurde (ISO/IEC 7816-3), und zum anderen das $T = 1$ Protokoll, das 1992 Eingang als Anhang in eine internationale Norm fand (damals in ISO/IEC 7816-3 Amd. 1 und jetzt in ISO/IEC 7816-3). Das stark an $T = 1$ angelehnte Vollduplex-Übertragungsprotokoll $T = 2$ befindet sich zurzeit in der Phase der Definition und wird in einigen Jahren als internationale Norm zur Verfügung stehen.

In Deutschland wurde an den weit verbreiteten Kartentelefonen noch ein drittes Protokoll namens $T = 14$ verwendet, das in einem Firmenstandard der Telekom festgelegt ist.

Die durch die Übertragungsprotokolle transportierten Dateneinheiten bezeichnet man als TPDU (*transmission protocol data unit*). Sie sind sozusagen protokollabhängige Container, die Daten von und zur Chipkarte transportieren. In ihnen sind dann die eigentlichen Anwendungsdaten eingebettet.

Zusätzlich zu den technisch aufwendigen Übertragungsprotokollen für Chipkarten existieren noch eine Reihe von sehr einfachen synchronen Protokollen für Speicherkarten. Diese finden z. B. bei Telefonkarten oder Krankenversichertenkarten ihren typischen Anwendungseinsatz. Sie haben allerdings keine Mechanismen zur Fehlerkorrektur und basieren auf einer fest verdrahteten Logik im Chip.

6.4.1 Synchrone Datenübertragung

Die synchrone Datenübertragung findet bei Chipkarten mit Mikrocontroller keine Verwendung, da diese nur asynchron mit dem Terminal kommunizieren. Sie ist aber der Standard bei Speicherkarten, die in sehr großen Stückzahlen, z. B. als vorbezahlte elektronische Geldbörsen bei Kartentelefonen, eingesetzt werden. Diese große Verbreitung ist auch der Grund, warum hier die synchrone Datenübertragung beschrieben ist.

Die synchrone Datenübertragung bei Speicherkarten ist sehr stark mit der Hardware des Chips verknüpft und auf eine größtmögliche Einfachheit hin ausgelegt. Es existiert auch keine Schichtentrennung im Übertragungsprotokoll oder eine logische Adressierung, sodass die Anwendung im Terminal direkt auf Speicheradressen des Chips zugreifen muss. Die dort enthaltenen Daten können durch das Protokoll physikalisch adressiert und dann gelesen oder geschrieben werden. Damit ist die eigentliche Datenübertragung auch noch zusätzlich mit der Funktionalität der Speicheradressierung und -verwaltung verknüpft.

Außerdem gibt es auch kein Verfahren, während der Datenübertragung Übertragungsfehler zu erkennen oder zu korrigieren. Im Übrigen treten Übertragungsfehler zwischen Chipkarte und Terminal sehr selten auf. Stellt die Anwendung im Terminal trotzdem einen Übertragungsfehler fest, muss sie den entsprechenden Bereich in der Speicherkarte nochmals lesen. Alle diese Einschränkungen ermöglichen es jedoch, mit einer geringen Anzahl von Logikgattern und hoher Geschwindigkeit Daten von der Karte zum Terminal und in umgekehrter Richtung zu übertragen.

Weil die synchrone Übertragung nur deshalb benutzt wird, um möglichst einfach, also mit einem minimalen Aufwand an Logikschaltkreisen, im Chip eine Datenübertragung zu ermöglichen, erhält man dadurch fast zwangsläufig eine starke Abhängigkeit zur verwendeten Hardware. So sind die synchronen Übertragungsprotokolle nicht einheitlich und zum Teil von Chip zu Chip sehr unterschiedlich. Lediglich der ATR ist standardisiert. Ein Terminal, das mit unterschiedlichen Speicherkarten kommunizieren will, benötigt also mehrere verschiedene Implementierungen von synchronen Übertragungsprotokollen.

Die exakte Bezeichnung für die Datenübertragung bei Speicherkarten lautet „zum Takt synchrone serielle Datenübertragung“. Damit wird erst klar, welche Randbedingungen mit dieser Kommunikationsart verbunden sind. Die Daten werden, genau wie im asynchronen Fall, seriell, also Bit für Bit, zwischen Karte und Terminal übertragen. Allerdings sind die zu übermittelnden Daten synchron zu einem zusätzlich übertragenen Takt. Damit entfällt die Notwendigkeit der Übertragung von Start- oder Stop-Informationen.

Im Fall einer einfachen Speicherkarte fehlen auch noch die Informationen für eine Fehlererkennung, es wird also auch kein Paritätsbit oder eine zusätzliche Prüfsumme mit übertragen. Dass die Wahrscheinlichkeit des Auftretens von Übertragungsfehlern gering ist, hat mit dem sehr niedrigen angelegten Takt zu tun. Er bewegt sich im Bereich von 10 kHz bis 100 kHz. Da pro Taktzyklus ein Bit übertragen wird, erreicht man beispielsweise bei einer Taktfrequenz von 20 kHz eine Übertragungsrate von 20 kBit/s. Allerdings ist die effektive Übertragungsrate etwas niedriger, da bei Speicherkarten noch zusätzliche Adressangaben mit übertragen werden.

Um die synchrone Datenübertragung bei Speicherkarten verständlich darzustellen, muss man vorab einige Grundlagen der Speicherkarten kennen. Diese Karten haben in ihrer einfachsten Form einen zweigeteilten Speicher, der aus einem nicht mehr änderbaren ROM und aus einem schreib- und löschbaren EEPROM besteht. Beide Bereiche sind bitweise adressierbar und können frei ausgelesen bzw. im Falle des EEPROM auch geschrieben und gelöscht werden.

Das Master-Slave-Verhalten bei Speicherkarten ist noch stärker ausgeprägt als bei Chipkarten mit Mikrocontrollern. Das Terminal übernimmt beispielsweise komplett die physikalische Adressierung des Speichers. Die Speicherkarte selber kann nur bestimmte Bereiche global gegen Löschen sperren. Die Steuerung dazu übernimmt eine dem Speicher vorgeschaltete festverdrahtete Logik, die ebenfalls die Datenübertragung verwaltet.

6.4.1.1 Protokoll für Telefonchips

Die Datenübertragung ist hier am Beispiel einer Telefonkarte mit einem Chip SLE 4403 der Firma Infineon dargestellt. Der Speicher dieses Bausteins ist bitorientiert, was bedeutet, dass alle Operationen auf einem einzelnen Bit ausgeführt werden. Andere Chiptypen können ein von dieser Beschreibung abweichendes Protokoll haben. Die Grundprinzipien der Datenübertragung sind jedoch bei allen synchronen Karten gleich.

Die Datenübertragung geschieht mit der Hilfe von drei Leitungen. Die bidirektionale Datenleitung kann sowohl vom Terminal als auch von der Karte zum Austausch von Daten in der Länge eines Bits genutzt werden. Die Taktleitung überträgt den vom Terminal erzeugten Takt zur Speicherkarte, welcher die Grundlage für die synchrone Datenübertragung ist. Die Dritte zur Übertragung notwendige Verbindung ist die Steuerleitung, die in Verbindung mit den Zuständen auf den beiden anderen Leitungen dann die eigentliche Funktion des Chips bestimmt.

Prinzipiell sind zur kompletten Ansteuerung einer Speicherkarte vier verschiedene Funktionen notwendig, die von der Logikschaltung im Chip decodiert werden müssen. Es sind dies die Funktionen Lesen, Schreiben, Löschen des Speichers und Verändern des Adresszeigers. Speicherkarten besitzen einen globalen Adresszeiger, mit dem alle Bereiche bitweise innerhalb des Speichers adressiert werden können. Erreicht der Adresszeiger das obere Ende des Speichers, läuft er nach null über. Bei einem bitorientierten Aufbau des Chips zeigt er dann also wieder auf das erste Bit im Speicher. Eine Funktion der synchronen Datenübertragung ist es, diesen Zeiger auf einen Anfangswert zurückzusetzen, der im Regelfall null ist.

Die nächste Funktion ist es, Daten aus dem Speicher zu lesen. Schreiben und Löschen von Bits im EEPROM ergänzen die beiden vorherigen Funktionen. Das Löschen und damit die Schaffung der Möglichkeit eines Neubeschreibens ist natürlich bei Telefonkarten gesperrt. Sonst könnten diese ja immer wieder aufgeladen werden.

Rücksetzen des Adresszeigers

Der Adresszeiger wird von der Anschaltlogik der Speicherkarte auf seinen Anfangswert null zurückgesetzt, wenn sich die Taktleitung und der Steuereingang gleichzeitig auf High-Pegel befinden. Der Steuerimpuls muss aber etwas länger anliegen als der High-Taktimpuls, um zu verhindern, dass die Adresse sofort um eins hochgezählt wird. Der Adresszeiger sollte nach jeder Anschaltsequenz der Karte auf seinen Ursprungswert zurückgesetzt werden, da er sonst auf eine undefinierte Adresse zeigen würde.

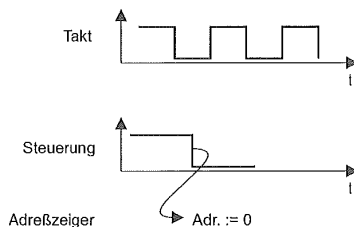


Bild 6.19 Das Rücksetzen des Adresszeigers auf die Adresse null.

Erhöhen des Adresszeigers und Lesen

Hat die Steuerleitung Low-Pegel und der Takt eine steigende Flanke, dann erhöht die interne Logik der Speicherkarte den Adresszeiger um eins. Mit der fallenden Flanke des Taktes wird der Inhalt der durch den Zeiger festgelegten Adresse auf die Datenleitung gelegt. Erreicht der Adresszähler seinen Maximalwert, der abhängig von der Größe des Kartenspeichers ist, läuft er auf null über und fängt somit wieder von ganz vorne bei null an.

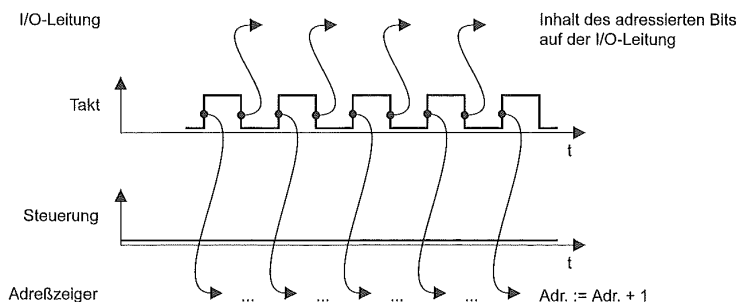


Bild 6.20 Das Erhöhen des Adresszeigers und Lesen von Daten von einer Adresse.

Schreiben auf eine Adresse

Steht der Adresszeiger auf einem beschreibbaren EEPROM-Bereich, dann kann der auf der Datenleitung anliegende Wert mit einem High-Pegel an der Steuerleitung während eines Low-Pegels des Taktes in das EEPROM geschrieben werden. Die Länge des Schreibvor-

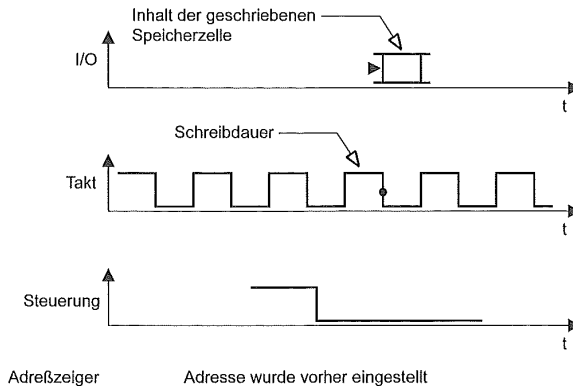


Bild 6.21 Das Schreiben eines Bits auf eine Adresse im EEPROM-Speicher.

gangs wird durch die Dauer des unmittelbar nachfolgenden Taktimpulses bestimmt. Am Datenausgang erscheint beim korrekten Schreiben der Inhalt der geschriebenen Speicherzelle.

Löschen von Bytes

Ein Teil des EEPROM-Speichers einer typischen Telefonkarte ist immer als mehrstelliger Oktalzähler organisiert. Muss nun in diesem Oktalzähler ein Byte gelöscht werden, da ein Übertrag auf die nächste Stelle erfolgte, dann stellt dies die Logikschaltung sicher. Deshalb ist das Löschen eines Bytes im Speicher etwas aufwendiger. Die Vorgehensweise ist dabei folgendermaßen: Schreibt man zweimal hintereinander auf ein Bit eines Bytes dann löscht die Hardwarelogik des Chips automatisch das dazugehörige niederwertige Byte. So ist sichergestellt, dass ein Übertrag auf die höhere Stelle erfolgte, und die niedrigere ist damit ohne eine Betrugsmöglichkeit gelöscht.

Die vier beschriebenen Zugriffsarten können von Chip zu Chip und auch von Hersteller zu Hersteller differieren. Eine andere Art der Datenübertragung, die jedoch standardisiert ist, findet man im I²C-Bus wieder. Viele der neueren Speicherkarten verwenden diesen Bus für die Kommunikation mit einem Terminal. Dies hat natürlich den Vorteil, dass verschiedene Bausteine von unterschiedlichen Herstellern in einem System parallel zueinander benutzbar sind. Probleme mit der Handhabung von mehreren Übertragungsprotokollen zur Karte fallen weg, da alle Chips auf der Übertragungsschnittstelle kompatibel zueinander sind.

6.4.1.2 I²C-Bus

Da serielle und taktsynchrone Datenübertragungsprotokolle unkompliziert und vielseitig einsetzbar sind, werden sie relativ häufig benutzt. Seit dem Jahr 1990 gibt es von der Firma Philips Bausteine mit der Eigenentwicklung I²C-Bus. Dieser Bus, dessen Abkürzung I²C für inter-integrated circuit steht, basiert auf einer seriellen und bidirektionalen Datenleitung und einer seriellen Taktleitung. Für diesen Bus ist sowohl die Hardware - also die beiden Leitungen - als auch die Software im Sinne des Formats für die Datenübertragung definiert. Jeder Baustein im Bus kann die Kontrolle über den Bus übernehmen und Anfragen an weitere an den Bus angeschlossene Bausteine senden.

Weil Speicherkarten ebenfalls taktsynchron angesteuert werden, hat sich sehr schnell der I²C-Bus in der Chipkartenwelt etabliert. Es gibt dafür mittlerweile die unterschiedlichsten Speicherbausteine für den Einbau in Chipkarten. Im folgenden Beispiel wird als Grundlage der Speicherchip ST24C04 von ST Microelectronics verwendet. Dieser Chip hat einen wahlfrei schreib- und lesbaren EEPROM-Speicher der Größe 512 Byte. Die Zeitberechnung für die Programmierung des EEPROMs wird intern vom Chip durchgeführt, muss also nicht von außen gesteuert werden.

Der I²C-Bus setzt als Hardware eine Zweidrahtleitung vom Terminal zur Chipkarte voraus. Die SCL-Leitung (*serial clock*) überträgt den bis 100 kHz betragenden Takt. Daraus resultiert eine für Chipkarten verhältnismäßig hohe Datenübertragungsgeschwindigkeit bis zu 100 kBit/s. Die andere Leitung, SDA (*serial data*), wird bidirektional zum Datenaustausch zwischen Chipkarte und Terminal benutzt. Die SDA-Leitung wird im Terminal mit einem Pull-Up-Widerstand mit der Versorgungsspannung Vcc verbunden. Die beiden Kommunikationspartner können diese Leitung nur auf Masse legen. Das Senden eines High-Pegels erfolgt also passiv, indem der Sender seinen Ausgang auf einen hochohmigen Zustand (Tri-State) schaltet und der Pull-Up-Widerstand die SDA-Leitung auf den Pegel der Versorgungsspannung hochzieht.

Im Zusammenhang mit Chipkarten ist das Terminal immer der Master des I²C-Busses und die Karte immer der Slave. Die Datenübertragung beim I²C-Bus findet grundsätzlich mit Paketen zu einem Byte statt. Das höherwertige Bit des Bytes (Bit 8) wird als Erstes übertragen. Jede Datenübertragung auf der SDA-Leitung wird durch ein Startsignal eingeleitet und mit einem Stoppsignal abgeschlossen. Das Startsignal erhält man durch eine fallende Flanke während eines High-Pegels der SCL-Leitung. Eine steigende Flanke während eines High-Pegels hingegen zeigt das Stoppsignal an. Jeder Empfang eines Bytes muss vom Empfänger mit einem Quittungssignal bestätigt werden. Dazu legt er die SDA-Leitung für einen Taktimpuls auf Masse.

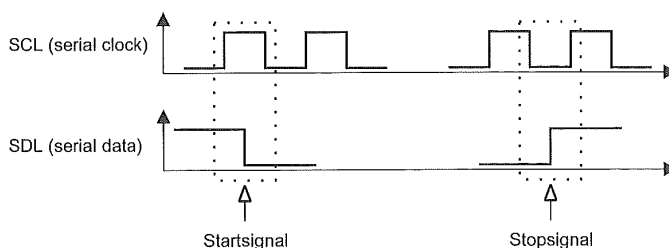


Bild 6.22 Das Start- und Stoppsignal beim I²C-Bus.

Nach dem Start der Kommunikation sind die ersten 7 Bit des ersten gesendeten Bytes die Adresse des Empfängers. Im Falle des hier dargestellten Beispiels ist aus Vereinfachungsgründen für die Adresse der binäre Wert 1010 000_x angenommen. Dieser kann natürlich für die verschiedenen Chiptypen unterschiedlich sein und lässt sich bei manchen Speicherbausteinen innerhalb gewisser Grenzen auch frei einstellen. Das letzte Bit in der Adressangabe zeigt dem Empfänger an, ob Daten geschrieben oder gelesen werden sollen. Der Wert 1 symbolisiert die Leseoperation, 0 hingegen die Schreiboperation.

Anhand der folgenden Beispiele wird die generelle Funktionsweise des I²C-Busses bei Chipkarten verdeutlicht.

Lesen von einer Adresse

Müssen Daten aus dem EEPROM-Speicher der Chipkarte gelesen werden, so gibt es dafür verschiedene Zugriffsarten. In der hier dargestellten wird ein einzelnes Byte ausgelesen. Es existiert aber auch die Möglichkeit, mehrere Bytes hintereinander aus dem Chip zu lesen.

Eingeleitet wird die Lesesequenz durch das Startsignal. Im Anschluss daran folgt die Adresse der Chipkarte, wobei das Zeichen für die Schreiboption gesetzt ist. Dies zeigt der Karte an, dass die nun folgenden Daten vorübergehend in einem internen Puffer aufbewahrt werden müssen. Dieser Puffer ist nichts anderes als ein byteorientierter Adresszeiger auf den EEPROM-Speicher. Nachdem die Chipkarte dieses erste Byte empfangen hat, sendet sie ein Quittungssignal aus. Dazu legt sie für die Zeitdauer eines Taktes die SDA-Leitung auf Masse. Nun folgt vom Terminal die Adresse im EEPROM der Chipkarte, die gelesen werden soll. Die Chipkarte wiederum bestätigt den Empfang. Anschließend sendet das Terminal ein Startsignal und die Adresse der Chipkarte mit gesetztem Lesesignal. Daraufhin sendet die Chipkarte die über den Adresszeiger adressierten Daten an das Terminal. Den Empfang der Daten muss das Terminal nicht bestätigen. Es sendet abschließend nur noch ein Stoppsignal zur Chipkarte. Damit ist die komplette Lesesequenz für ein einzelnes Byte abgeschlossen.

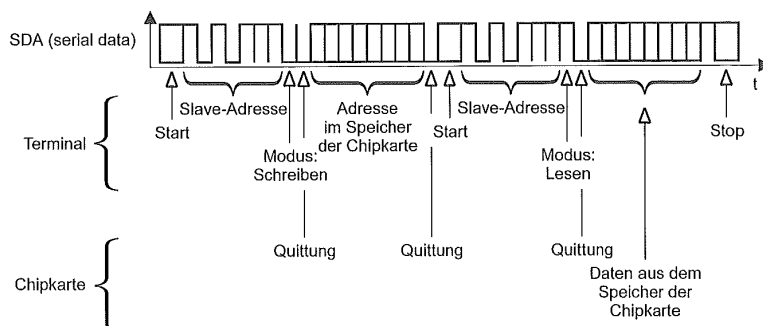


Bild 6.23 Das wahlfreie Lesen eines Bytes in einer Speicherkarte mit dem I²C-Bus.

Schreiben auf eine Adresse

Analog dem Lesen von Daten aus dem EEPROM der Chipkarte gibt es auch für das Schreiben verschiedene Modi. Im Folgenden ist der einfachste davon dargestellt, mit dem ein einzelnes Byte wahlfrei in den Speicher geschrieben werden kann.

Die Sequenz beginnt wiederum mit einem Startsignal vom Terminal. Es folgt die Adresse der Chipkarte mit gesetzter Schreiboption. Die Chipkarte bestätigt den Empfang und erhält daraufhin vom Terminal die Adresse im EEPROM, auf die geschrieben werden soll.

Auch dies wird von der Chipkarte bestätigt, welche anschließend die zu schreibenden Daten erhält. Nach der dritten Bestätigung des Erhalts von Daten durch die Chipkarte sendet das Terminal ein Stoppsignal. Daraufhin beginnt die Chipkarte das erhaltene Datenbyte in das EEPROM zu schreiben, wozu keine Zeitsteuerung von außen notwendig ist. Nun erst ist die komplette Schreibsequenz abgeschlossen und das übergebene Byte in das EEPROM geschrieben.

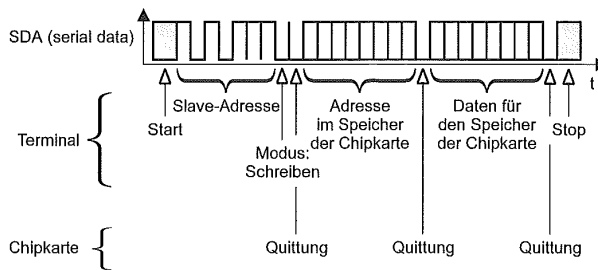


Bild 6.24 Das wahlfreie Schreiben eines Bytes in eine Speicherkarte mit dem I²C-Bus.

6.4.2 Übertragungsprotokoll T = 0

Dieses Übertragungsprotokoll wurde schon in der Anfangsphase der Chipkarten-Entwicklung in Frankreich verwendet und war auch das erste international genormte Protokoll für Chipkarten. Es ist in den ersten Jahren der Chipkartentechnik entstanden und daher auf minimalen Speicherbedarf und maximale Einfachheit ausgelegt. Dieses Protokoll wird weltweit in der GSM-Karte verwendet und hat aus diesem Grund momentan den größten Verbreitungsgrad aller Chipkarten-Protokolle. Genormt ist T=0 in ISO/IEC 7816-3. Weitere kompatible Spezifikationen befinden sich in der GSM 11.11, TS 102.221 und dem EMV-Standard.

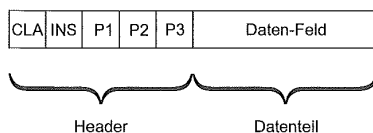


Bild 6.25 Der Aufbau eines Kommandos bei T = 0.

Das T=0 Protokoll ist byteorientiert, was bedeutet, dass die kleinste Einheit, die das Protokoll bearbeitet, ein einzelnes Byte ist. Die übertragenen Dateneinheiten setzen sich aus einem Header mit Class-, Instruction- und drei Parameterbytes zusammen. Danach folgt optional ein Datenteil. Im Gegensatz zu den in ISO/IEC 7816-4 genormten APDUs existiert als Längenangabe nur der Parameter P3. Er gibt entweder die Länge der Daten des Kommandos oder die der Antwort an und ist nach ISO/IEC 7816-3 ebenfalls genormt.

Durch die Byteorientierung von T=0 direkt muss nach einem erkannten Übertragungsfehler das nicht korrekt empfangene Byte nochmals angefordert werden. Im Gegensatz dazu ist im Fehlerfall bei Blockprotokollen der gesamte Block (d. h. eine Sequenz von Bytes) nochmals zu senden. Die Erkennung von Übertragungsfehlern erfolgt bei T=0 ausschließlich mithilfe des Paritätsbits, das jedem übertragenen Byte nachgestellt ist.

Erkennt der Empfänger einen Übertragungsfehler, dann muss er, nachdem das erste Bit der Guardtime des fehlerhaften Bytes zur Hälfte verstrichen ist, die I/O-Leitung für die Dauer einer etu auf low Pegel setzen. Damit signalisiert er dem Kommunikationspartner, dass dieser das zuletzt gesendete Byte nochmals senden muss. Dieser Mechanismus der „Byte-Repetition“ ist sehr einfach und hat den Vorteil, dass selektiv nur das nicht korrekt empfangene Byte wiederholt werden muss.

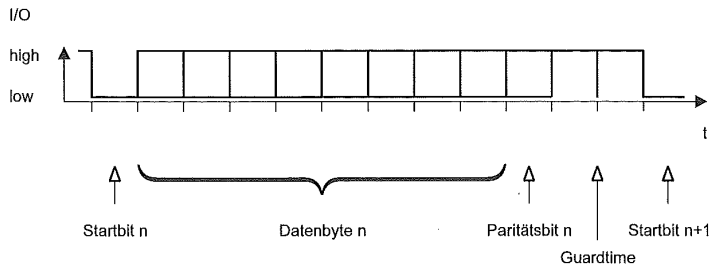


Bild 6.26 Darstellung eines fehlerfrei übertragenen Bytes bei $T = 0$ auf der I/O-Schnittstelle.

Leider ist das Verfahren mit einem gravierenden Nachteil verbunden. Die meisten Schnittstellenbausteine behandeln die Zeitdauer von einem etu als die kleinste detektierbare Einheit, sodass sie die nach einem halben Stopbit auf Low-Pegel gesetzten I/O-Leitung nicht erkennen können. Deshalb eignen sich Standard-Schnittstellen-Bausteine nicht für das $T = 0$ Protokoll. Wird jedoch jedes Bit einzeln per Software empfangen, so ist dies kein Problem mehr.

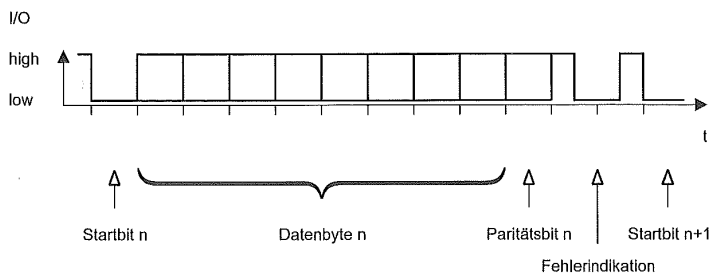


Bild 6.27 Signalisierung eines Übertragungsfehlers bei $T = 0$ durch einen Low-Pegel auf der I/O-Schnittstelle während der Guardtime.

Das $T = 0$ Protokoll würde auch noch die Möglichkeit vorsehen, eine externe Programmierspannung für das EEPROM bzw. EPROM der Chipkarte ein- und auszuschalten. Dazu addiert man den Wert eins auf das empfangene Instruction Byte und sendet es als Acknowledge Byte an das Terminal zurück. Dies ist im Übrigen auch der Grund, warum nur geradzahlige Instruction Bytes erlaubt sind, denn sonst würde der vorangehend beschriebene Mechanismus nicht mehr funktionieren. Das Schalten einer externen Programmierspannung ist jedoch technisch überholt, da ausnahmslos alle Chipkarten-Mikrocontroller die Programmierspannung auf dem Chip erzeugen. Deshalb wird hier im weiteren Verlauf auch nicht mehr darauf eingegangen.¹⁰

Um den Ablauf einer $T = 0$ Kommando-Antwort-Sequenz zu zeigen, nehme man an, dass das Terminal ein Kommando mit Datenteil zur Chipkarte sendet und diese daraufhin Daten und einen Antwortcode zurückschickt.

¹⁰ Ein Änderungsvorschlag für die Überarbeitung der ISO/IEC 7816-3 sieht vor, dass die Steuerung einer externen Programmierspannung über das Instruction Byte in Zukunft nicht mehr vorgesehen ist. Damit würde die Anzahl der möglichen Kommandos verdoppelt.

Das Terminal sendet als Erstes einen 5 Byte langen Kommando-Header, bestehend aus Class-, Instruction-, P1-, P2- und P3-Byte, zur Karte. Wird dieser korrekt empfangen, so sendet diese als Procedure-Byte (PB) ein Acknowledge (ACK) zurück. Codiert ist dieses Acknowledge als das empfangene Instruction Byte. Nachdem das Terminal das Procedure-Byte empfangen hat, sendet es exakt die Anzahl von Datenbytes, die im P3-Byte angezeigt wurden. Damit hat die Karte das komplette Kommando empfangen, kann es abarbeiten und eine Antwort erzeugen.

Enthält die Antwort neben dem 2 Byte langen Returncode auch Daten, so teilt dies die Karte dem Terminal mit einem speziellen Returncode und der Anzahl der Daten in SW2 mit. Nach Empfang dieser Antwort sendet das Terminal ein so genanntes GET RESPONSE-Kommando, das nur aus einem Kommando-Header mit der Anzahl der zu übertragenden Daten an die Karte besteht. Diese sendet nun die nach dem ersten Kommando erzeugten Daten in der angeforderten Länge und mit dem dazugehörigen Returncode an das Terminal. Damit ist eine Kommandosequenz abgeschlossen.

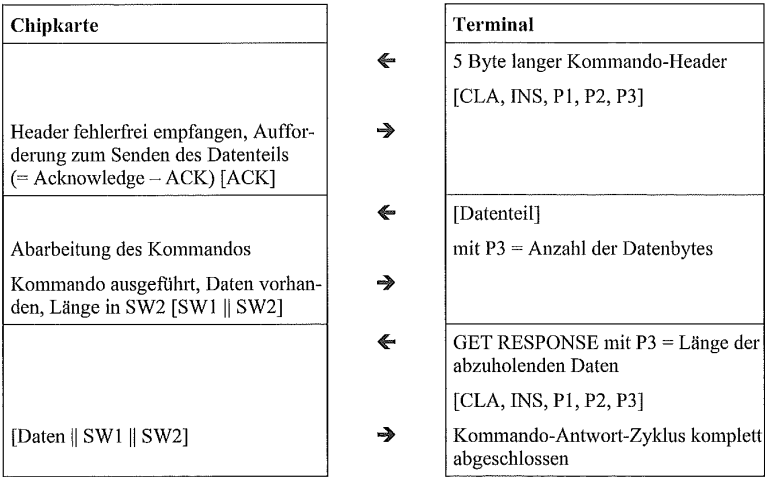


Bild 6.28 Ein typischer Kommunikationsablauf bei T = 0 mit Daten sowohl im Kommando als auch in der Antwort (Case 4 Kommando, wie z. B. MUTUAL AUTHENTICATE).

Wird ein Kommando zur Karte gesendet, und diese erzeugt nur einen Returncode ohne Datenteil, so entfällt der Teil mit GET RESPONSE. Da für diese Aktion des Abholens von Daten eines vorherigen Kommandos ein zusätzliches Kommando der Anwendungsschicht notwendig ist, herrscht natürlich keine exakte Trennung der Protokoll-Schichten mehr. Man muss Kommandos der Anwendungsschicht (GET RESPONSE) zur Unterstützung der Leitungsschicht benutzen, was einige Auswirkungen auf die jeweilige Anwendung hat. Obige Abläufe mögen auf den ersten Blick kompliziert erscheinen, deshalb sind sie in Bild 6.30 nochmals grafisch dargestellt.

Der maximale zeitliche Abstand zwischen den Startflanken zweier aufeinander folgender Bytes wird als work waiting time bezeichnet. Die wird work waiting ist im ATR im Datenelement TC2 codiert.

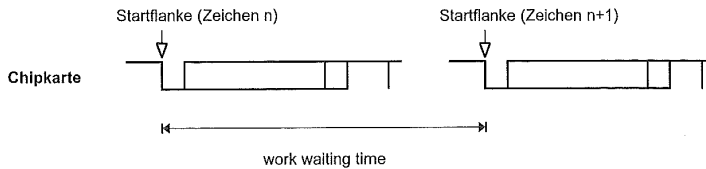
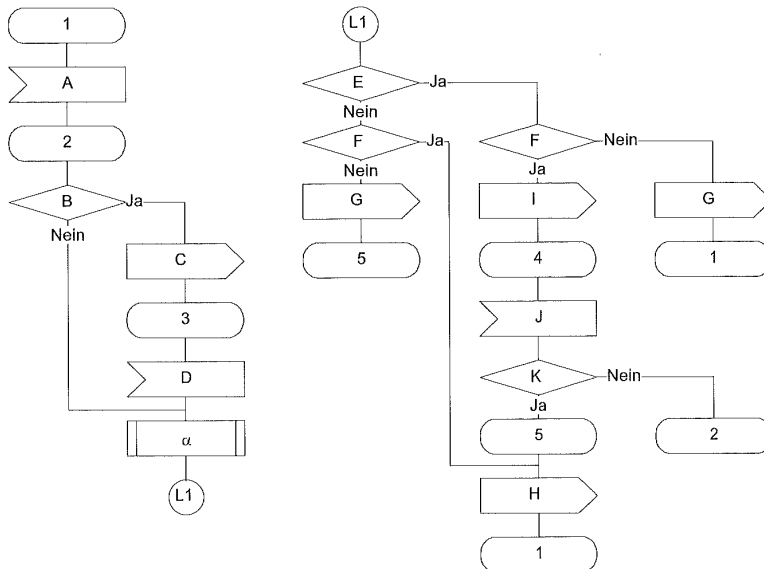


Bild 6.29 Die Definition der Work Waiting Time.

Bild 6.30 Der Zustandsautomat der Chipkarte für den Kommunikationsablauf bei $T = 0$ ohne Fehlerbehandlung.

- α – Abarbeitung des Kommandos
- 1 – Grundzustand
- 2 – Header mit CLA, INS, P1, P2, P3 empfangen
- 3 – Warte auf Datenteil (P3 = Anzahl der Bytes)
- 4 – Warte auf ein Kommando (Header mit CLA, INS, P1, P2, P3)
(P3 = Anzahl der Antwortdaten)
- 5 – SW1, SW2 gesendet / GET RESPONSE empfangen
- A – Empfange Header (5 Byte)
- B – Datenteil vorhanden (P3 $\neq$ 0)?
- C – Datenteil vorhanden, sende Prozedure-Byte zum Terminal
- D – Empfange Datenteil (P3 = Anzahl der Bytes)
- E – War ein Datenteil im Kommando vorhanden (d. h. C und D durchlaufen)?
- F – Sind Antwortdaten vorhanden (d. h. kein Fehler aufgetreten)?
- G – Sende SW1 und SW2
- H – Sende vorhandene Antwortdaten und SW1, SW2
- I – Sende SW1 und SW2 (SW2 = Anzahl der Antwortdaten)
- J – Empfange ein Kommando (Header = 5 Byte)
- K – Ist das empfangene Kommando ein GET RESPONSE?

Die Guardtime, die auch Schutzzeit heißt, hat vor allem die Funktion, die einzelnen Bytes während der Übertragung zu trennen. Dadurch hat sowohl der Sender als auch der Empfänger mehr Zeit für die Funktionen des Übertragungsprotokolls.

Sendet die Chipkarte als Procedure-Byte den Null-Wert (= '60') an das Terminal zurück, so hat dies keine Auswirkungen auf den eigentlichen Protokollablauf, das Terminal wird jedoch mit dieser Prozedur darüber informiert das die Chipkarte noch das als letztes gesendete Kommando abarbeitet. Man kann dieses Senden des Null-Wertes jedoch als eine Art Wartezeitverlängerung (*waiting time extension* – WTX) benutzen, obwohl dies in dieser Form nicht genormt ist.

Das T = 0 Protokoll ermöglicht der Karte nach Empfang des Headers, die Bytes im Datenteil einzeln zu empfangen. Dazu muss diese lediglich als Procedure-Byte das invertierte Instruction-Byte zum Terminal senden, das daraufhin nur ein einzelnes Datenbyte aussendet. Beim nächsten Procedure-Byte der Karte folgt dann das nächste Datenbyte. Dieses byteweise Senden kann so lange fortgesetzt werden, bis die Karte alle Bytes des Datenteils empfangen hat oder als Procedure-Byte das nicht invertierte Instruction-Byte zum Terminal sendet. Daraufhin schickt das Terminal alle verbliebenen Bytes des Datenteils zur Karte, und diese hat somit das vollständige Kommando empfangen.

An dieser Stelle gibt es zwei Inkompatibilitäten zwischen GSM 11.11 und ISO/IEC 7816-3. Nach GSM wird ein mögliches GET RESPONSE mit SW1 = '9F' angefordert und nach ISO/IEC üblicherweise mit '61'¹¹. SW2 beinhaltet jeweils die Anzahl der abzuholenden Daten.

Der zweite Unterschied zwischen den beiden Normen betrifft die Art der Abholung der Daten mit GET RESPONSE. Die oben beschriebene Art entspricht GSM und vertritt damit die weltweit größte Chipkarten-Anwendung. Nach ISO/IEC können mit GET RESPONSE eine bestimmte Anzahl Daten abgeholt werden wobei es keinen sich weiterschaltenden Zeiger gibt, sodass ein Datenpaket nach dem Anderen angefordert werden kann. Bei ISO/IEC wird bei GET RESPONSE immer mit dem ersten Byte begonnen. Diese beiden Inkompatibilitäten können durch entsprechende Software im Terminal problemlos entschärft werden. Das Kritische dabei ist nur das Wissen, dass sie existieren.

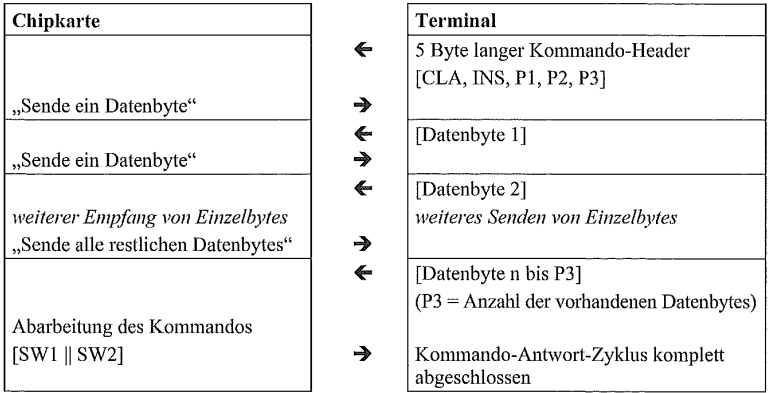


Bild 6.31 Der Einzelbyte-Empfang bei T = 0 (Case 3 Kommando wie z. B. bei UPDATE BINARY).

¹¹ Ein Vorschlag zur Revision der ISO/IEC 7816-3 sieht für die Zukunft eine Erweiterung vor, sodass nach ISO/IEC beide Werte erlaubt wären.

Der Anwender interessiert sich bei einem Übertragungsprotokoll letztendlich vor allem für die Datenübertragungsgeschwindigkeit sowie die Fehlererkennungs- und Korrekturmechanismen.

Zum Senden eines Bytes, bestehend aus 8 Bit, müssen 12 Bit übertragen werden, da man 1 Startbit, 1 Paritätsbit und 2 t_u für die Guardtime berücksichtigen muss. Unter der Annahme von 3,5712 MHz Takt und des Teilers 372 dauert damit die Übertragung von 1 Byte die Zeit von $12 \cdot 1 \text{ etu} = 1,25 \text{ ms}$.

Folgende Tabelle zeigt für einige typische Kommandos die benötigte Zeit zur Datenübertragung:

Tabelle 6.33 Übertragungsraten bei $T = 0$ und verschiedenen Kommandos (Cmd: Kommando, Rsp: Antwort) mit den folgenden Annahmen: Takt: 3,5712 MHz, Teiler: 372, 2 Stopbits, und jeweils 8 Byte Daten pro Kommando.

Kommando	Anzahl der Nutzdaten	Anzahl der Protokoll- daten	Dauer der Daten- übertragung
READ BINARY	Cmd: 5 Byte Rsp: 2 + 8 Byte	---	18,75 ms
UPDATE BINARY	Cmd: 5 + 8 Byte Rsp: 2 Byte	---	18,75 ms
ENCRYPT	Cmd: 5 + 8 Byte Rsp: 2 + 8 Byte	Cmd: 5 Byte Rsp: 2 Byte	37,50 ms

Die Übertragungsrate sinkt natürlich, sobald Übertragungsfehler auftreten. Allerdings wirkt sich hier der Mechanismus der Einzelbytewiederholung sehr positiv aus, da selektiv nur das nicht korrekt empfangene Byte wiederholt gesendet wird.

Der Fehlererkennungsmechanismus des $T = 0$ Protokolls umfasst lediglich eine Paritätsprüfung am Ende jeden Bytes. Damit können 1-Bit-Fehler sicher erkannt werden, 2-Bit-Fehler lassen sich damit schon nicht mehr detektieren. Geht während der Übertragung vom Terminal zur Karte ein Byte verloren, so führt dies zu einer Endlosschleife (*dead lock*) in der Karte, da diese auf eine bestimmte Anzahl von Bytes wartet und keine Möglichkeit eines zeitgesteuerten Abbruchs hat. Der einzige praktikable Ausweg für das Terminal, diese Sackgasse in der Kommunikation zu beseitigen, ist ein Reset der Karte mit einem kompletten Neuaufbau der Verbindung.

Ganz ähnlich verhält es sich, wenn das Terminal auf mehr Daten wartet als die Chipkarte sendet. Auch dies würde unweigerlich zu einer Verklemmung (*dead lock*) führen. Aus diesem Grund haben manche Realisierungen von $T = 0$ in Terminals einen Timer, der nach einer einzustellenden Maximalzeit einen Kommunikationsabbruch durch das Terminal initiiert. Der dabei verwendete Mechanismus verhält sich analog dem der BWT (*block waiting time*) bei $T = 1$. Er ist jedoch nicht genormt und abhängig von der jeweiligen Implementierung.

In der normalen Kommunikation verursacht die nicht ausreichende Trennung zwischen Leitungs- und Transportschicht keine allzu großen Auswirkungen. Die gut funktionierende Anwendung GSM ist das beste Beispiel dafür. Beim Einsatz von gesicherter Datenübertragung, kommt es indes sehr bald zu Problemen. Denn bei einem teilweise verschlüsselten

Header und einem vollständig verschlüsseltem Datenteil ist es durch den vorgegebenen Protokollablauf nicht mehr möglich, $T=0$ ohne größeren Aufwand zu unterstützen. Die Ursache dafür ist, dass ein unverschlüsseltes Instruction-Byte zur Verwendung als Procedure-Byte von $T=0$ benötigt wird. Dies wurde jedoch von der internationalen Normung erkannt und in den entsprechenden Normen für Secure Messaging berücksichtigt, sodass auch mit $T=0$ alle Spielarten von Secure Messaging möglich sind.

Wegen der nicht vorhandenen Schichtentrennung und der offensichtlichen Probleme bei einer schlechten Verbindung wird das $T=0$ Protokoll oft als veraltet betrachtet. Andererseits kommt es bei der Kommunikation zwischen Terminal und Chipkarte so gut wie nie zu Übertragungsfehlern. Die gute durchschnittliche Übertragungsgeschwindigkeit, der geringe Aufwand zur Implementierung und die große Verbreitung machen die wesentlichen Vorteile dieses Protokolls aus.

6.4.3 Übertragungsprotokoll $T=1$

Das Übertragungsprotokoll $T=1$ ist ein asynchrones Halbduplexprotokoll für Chipkarten. Die Basis ist dabei die internationale Norm ISO/IEC 7816-3. Andere Spezifikationsgrundlagen sind TS 102.221 und der EMV-Standard. Das $T=1$ Protokoll gehört zur Klasse der Blockprotokolle. Dies heißt, dass ein Block die kleinste Dateneinheit ist, die zwischen Chipkarte und Terminal übertragen werden kann.

Dieses Protokoll weist eine strenge Schichtentrennung auf und kann in das OSI-Referenzmodell als Leitungsschicht (*data link layer*) eingeordnet werden. Schichtentrennung heißt in diesem Zusammenhang auch, dass Daten, die für höhere Schichten, also beispielsweise die Anwendungsschicht, bestimmt sind, vollständig transparent von der Leitungsschicht verarbeitet werden können. Es ist nicht notwendig, dass weitere Schichten – außer der betroffenen – den Inhalt der übertragenen Daten interpretieren oder verändern müssen.

Gerade der Einsatz von gesicherter Datenübertragung (*secure messaging – SM*) erfordert es, dass eine Schichtentrennung eingehalten wird. Nur dann ist es ohne großen Aufwand und ohne Tricks möglich, Anwendungsdaten verschlüsselt über die Schnittstelle zu übertragen. Das $T=1$ -Protokoll ist momentan das einzige internationale Protokoll für Chipkarten, mit dem ohne Kompromisse eine gesicherte Datenübertragung in allen Variationen möglich ist.

Der Ablauf des Übertragungsprotokolls startet, nachdem die Chipkarte den ATR ausgesendet hat oder nachdem ein erfolgreicher PPS ausgeführt wurde. Den ersten Block sendet das Terminal aus, den darauf Folgenden die Chipkarte. Die Kommunikation wird dann mit wechselndem Senderecht in diesem Sinne weitergeführt.

Im Übrigen findet das $T=1$ -Übertragungsprotokoll nicht nur zwischen Chipkarten und Terminals seine Anwendung. Es gibt viele Terminals, bei denen es zur Kommunikation mit dem daran angeschlossenen Computer zur Übertragung der Anwendungs- und Steuerungsdaten eingesetzt wird.

Bei jedem Übertragungsprotokoll ist natürlich die resultierende Übertragungsgeschwindigkeit von besonderem Interesse. In der folgenden Tabelle sind einige typische Kommandos und die benötigte Zeit zur Datenübertragung aufgeführt:

Tabelle 6.34 Übertragungsraten bei T = 1 und verschiedenen Kommandos (Cmd: Kommando, Rsp: Antwort) und den folgenden Annahmen: Takt: 3,5712 MHz, Teiler: 372, XOR als Fehlererkennungscod, 2 Stopbits und jeweils 8 Byte Daten pro Kommando.

Kommando	Anzahl der Nutzdaten		Anzahl der Protokoll-daten	Dauer der Daten-übertragung
READ BINARY	Cmd:	5 Byte	Cmd: 4 Byte	28,75 ms
	Rsp:	2 + 8 Byte	Rsp: 4 Byte	
UPDATE BINARY	Cmd:	5 + 8 Byte	Cmd: 4 Byte	23,00 ms
	Rsp:	2 Byte	Rsp: 4 Byte	
ENCRYPT	Cmd:	5 + 8 Byte	Cmd: 4 Byte	38,75 ms
	Rsp:	2 + 8 Byte	Rsp: 4 Byte	

6.4.3.1 Blockaufbau

Die zu übertragenden Blöcke werden grundsätzlich für zwei verschiedene Zwecke benutzt. Einerseits zur transparenten Übertragung von anwendungsspezifischen Daten und andererseits für die Steuerdaten des Übertragungsprotokolls bzw. für die Behandlung von Übertragungsfehlern.

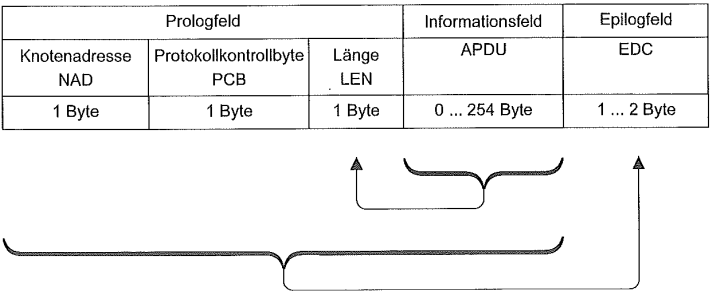


Bild 6.32 Der Aufbau eines T = 1 Übertragungsblocks.

Die Übertragungsblöcke setzen sich aus einem führenden Prologfeld, dem Informationsfeld und einem nachgestellten Epilogfeld zusammen. Prolog- und Epilogfeld sind obligatorisch, müssen also in jedem Fall gesendet werden. Das Informationsfeld ist im Gegensatz dazu optional und enthält die Daten für die Anwendungsschicht. Dies ist entweder eine Kommando-APDU für die Chipkarte oder eine Antwort-APDU von der Chipkarte.

Grundsätzlich unterscheidet man im T = 1-Protokoll drei verschiedene Blocktypen: Informationsblöcke, Empfangsbestätigungsblöcke und Systemblöcke: Die Informationsblöcke (I-Blöcke) werden zum transparenten Austausch von Daten der Anwendungsschicht genutzt. Der Empfangsbestätigungsblock (R-Block), der kein Informationsfeld besitzt, dient für positive oder negative Empfangsbestätigungen. Systemblöcke (S-Block) nutzt man für Steuerinformationen, die das Protokoll selbst betreffen. In Abhängigkeit von der Steuerinformation können sie ein Informationsfeld besitzen.

Prologfeld

Das Prologfeld besteht aus den drei Unterfeldern Knotenadresse (NAD), Protokollkontrollbyte (PCB) und Länge (LEN). Es hat eine Länge von drei Bytes und enthält grundlegende Steuerungs- und Anzeiginformationen für den aktuellen Übertragungsblock.

Knotenadresse NAD (*node address*)

Das erste Byte des Prologfeldes wird als Knotenadresse bzw. NAD-Byte bezeichnet. Es enthält die Ziel- und Quellknotenadresse des Blocks. Diese sind in je 3 Bit codiert. Wird die Adressierung nicht benutzt, so sind die jeweiligen Bits auf 0 gesetzt.

Des Weiteren ist aus Kompatibilitätsgründen zu älteren Mikrocontrollern eine Steuerung der EEPROM- bzw. EPROM-Programmierspannung vorgesehen. Diese Steuerung findet aber keine praktische Anwendung mehr, da seit langem alle Mikrocontroller für Chipkarten eine Ladungspumpe auf dem Chip haben.

Tabelle 6.35 Die Knotenadresse (NAD-Feld)

b8	b7	b6	b5	b4	b3	b2	b1	Bedeutung
X	...	...	...	X	...	...	...	Vpp-Steuerung
...	X	X	X	...	...	...	...	Zielknotenadresse DAD (<i>destination address</i>)
...	...	...	...	...	X	X	X	Quellknotenadresse SAD (<i>source adress</i>)

PCB-Feld (*protocol control byte*)

Das der Knotenadresse folgende Unterfeld ist das Protokollkontrollbyte, kurz PCB-Byte (*protocol control byte*) genannt. Wie der Name schon beschreibt, dient es zur Kontrolle und Steuerung des Übertragungsprotokolls. Damit steigt der Aufwand an notwendigen Codierungen. Das PCB-Feld codiert in erster Linie den Blocktyp sowie die dafür notwendigen Zusatzinformationen.

Tabelle 6.36 Das PCB-Feld im Falle eines I-Blocks

b8	b7	b6	b5	b4	b3	b2	b1	Bedeutung
0	...	...	...	...	...	...	...	Kennung für I-Block
...	N(S)	...	...	...	...	...	...	Sendefolgenummer
...	...	X	...	...	...	...	...	Folgedatenbit M
...	...	...	X	X	X	X	X	Reserviert

Tabelle 6.37 Das PCB-Feld im Falle eines R-Blocks

b8	b7	b6	b5	b4	b3	b2	b1	Bedeutung
1	0	...	...	...	...	...	...	Kennung für R-Block
...	...	0	N(R)	0	0	0	0	kein Fehler
...	...	0	N(R)	0	0	0	1	EDC-/Paritätsfehler
...	...	0	N(R)	0	0	1	0	Sonstiger Fehler

Tabelle 6.38 Das PCB-Feld im Falle eines S-Blocks

b8	b7	b6	b5	b4	b3	b2	b1	Bedeutung
1	1	...	...	...	...	...	...	Kennung für S-Block
...	...	0	0	0	0	0	0	Resynch Anfrage (nur vom Terminal)
...	...	1	0	0	0	0	0	Resynch Antwort (nur von der Chipkarte)
...	...	0	0	0	0	0	1	Anfrage zur Größenänderung des Informationsfeldes
...	...	1	0	0	0	0	1	Antwort zur Größenänderung des Informationsfeldes
...	...	0	0	0	0	1	0	Abbruch Anfrage
...	...	1	0	0	0	1	0	Abbruch Antwort
...	...	0	0	0	0	1	1	Anfrage zur Wartezeitverlängerung (nur von der Chipkarte)
...	...	1	0	0	0	1	1	Antwort zur Wartezeitverlängerung (nur vom Terminal)
...	...	1	0	0	1	0	0	Vpp Fehler Antwort (nur von der Chipkarte)

LEN-Feld (*length field*)

Das 1 Byte lange LEN-Feld zeigt in hexadezimaler Form die Länge des Informationsfeldes an. Der Wert des LEN-Feldes kann dabei von '00' bis 'FE' sein. Die Codierung 'FF' ist für zukünftige Erweiterungen vorgesehen und sollte daher momentan noch nicht benutzt werden.

Informationsfeld

Das Informationsfeld dient im Falle eines I-Blocks als Container für die Daten der Anwendungsschicht (OSI-Schicht 7). Der Inhalt dieses Feldes wird vollständig transparent übertragen. Das bedeutet, dass der Inhalt vom Übertragungsprotokoll ohne Analyse oder Auswertung direkt weitergeleitet wird.

Im Falle eines S-Blocks werden in diesem Feld Daten für das Übertragungsprotokoll übertragen. Dies ist jedoch die einzige Ausnahme, bei der der Inhalt des I-Feldes von der Übertragungsschicht verwendet wird.

Die Größe des I-Feldes kann sich nach ISO zwischen '00' und 'FE' (254) Byte bewegen. Der Wert 'FF' (255) ist für die zukünftige Verwendung durch ISO reserviert. Terminal und Karte können unterschiedlich große I-Felder haben. Der voreingestellte Wert der I-Feld-Länge für das Terminal beträgt 32 Byte (*information field size for the interface device – IFSD*) und kann mit einem speziellen S-Block verändert werden. Für die Chipkarte gilt ebenfalls dieser voreingestellte Wert von 32 Byte (= *information field size for the card – IFSC*), doch kann man ihn durch Angabe im ATR abändern.¹² In der Praxis liegt die Länge des I-Feldes sowohl bei den Terminals als auch bei den Chipkarten im Bereich zwischen 50 Byte und 254 Byte.

Epilogfeld

Dieses am Ende des Blocks übertragene Feld enthält einen Fehlererkennungscode über alle vorangegangenen Bytes des Blocks. Die Berechnung geschieht entweder mit einem

¹² Datenelement im ATR: TAI (i > 2)

Längsprüfungscode (*longitudinal redundancy check – LRC*) oder mit einem zyklischen Redundanzprüfungscode (*cyclic redundancy check – CRC*). Welcher dieser beiden Fehlererkennungscode verwendet wird, muss in den interface characters des ATR angegeben werden. Ist dort keine Angabe enthalten, so ist implizit die LRC-Methode vereinbart. Im anderen Fall ist die CRC-Berechnung nach ISO 3309 vorzunehmen. Das dort benützte Teilerpolynom $G(x) = x^{16} + x^{12} + x^5 + 1$ ist identisch mit der CCITT Empfehlung V.41¹³. Die beiden Fehlererkennungscode können nur zu einer Fehlerdetektion genutzt werden, es ist mit ihnen nicht möglich, eine Fehlerkorrektur im Block vorzunehmen.

Der ein Byte lange Längsprüfungscode berechnet sich durch die XOR-Verknüpfung aller vorangehenden Bytes im Block. Diese Berechnung lässt sich sehr schnell durchführen und ist in der Implementierung wenig codeintensiv. Normalerweise führt man sie on-the-fly während des Sendens bzw. Empfangens durch. Sie ist in praktisch allen T = 1 Implementierungen als Standard vorhanden.

Die Verwendung des CRC-Verfahrens zur Erzeugung eines Fehlererkennungscode ergibt eine weit höhere Wahrscheinlichkeit der Fehlererkennung als die relativ primitive XOR-Prüfsumme. Allerdings findet dieses Verfahren in der Praxis momentan keine Verwendung, da sich weltweit die XOR-Prüfsumme fest eingebürgert hat. Beim CRC-Verfahren muss das Epilogfeld auf zwei Byte erweitert werden, was die Übertragungsrate senkt.

6.4.3.2 Sendefolge-/Empfangsfolgezähler (*send sequence counter/receive sequence counter*)

Jeder Informationsblock im T = 1-Protokoll ist mit einer Sendefolgenummer versehen, die aus nur einem Bit besteht und sich im PCB-Byte befindet. Sie wird modulo 2 hochgezählt, d. h. sie nimmt abwechselnd die Werte 0 und 1 an. Der Sendefolgezähler hat auch noch die Bezeichnung N(S) und hat den Wert null als Anfangswert beim Start des Protokolls. Die Zähler beim Terminal und in der Chipkarte werden unabhängig voneinander erhöht.

Der Sendefolgezähler hat vorwiegend den Zweck, einen fehlerhaft empfangenen Block nochmals anzufordern, da sich über N(S) die einzelnen Informationsblöcke jeweils eindeutig adressieren lassen.

6.4.3.3 Wartezeiten

Um Sendern und Empfängern bei der Datenübertragung genau festgelegte Mindest- und Maximalzeiträume für diverse Aktionen einzuräumen und um im Fehlerfall über einen definierten Abbruch eine Blockade des Übertragungsprotokolls zu verhindern, wurden verschiedene Wartezeiten definiert. Diese sind in der Norm alle mit Standardwerten vorbelegt, doch um die Übertragungsgeschwindigkeit zu maximieren, ist es möglich, diese entsprechend zu verändern und in den Specific Interface Characters des ATR dann anzuzeigen.

Zeichenwartezeit CWT (*character waiting time*)

Die Zeichenwartezeit CWT ist definiert als die maximale Zeitspanne zwischen den Startflanken zweier aufeinander folgender Zeichen innerhalb eines Blocks. Der Empfänger startet also bei jeder Startflanke eine Countdown-Uhr. Diese Uhr hat als Startwert die Zeichenwartezeit. Läuft die Uhr ab und wurde währenddessen noch keine Startflanke eines

¹³ Siehe auch Abschnitt 4.5.2 „CRC-Prüfsummen“.

neuen Zeichens detektiert, geht der Empfänger davon aus, dass der Übertragungsblock vollständig empfangen wurde.

Das „CWT-Empfangskriterium“ kann daher auch generell als Blockendeerkennung verwendet werden. Dies reduziert die Geschwindigkeit der Datenübertragung erheblich, da sich jeder Block um die Zeitspanne der CWT verlängert. Besser geeignet ist daher die Erkennung des Blockendes durch das Zählen der empfangenen Bytes.

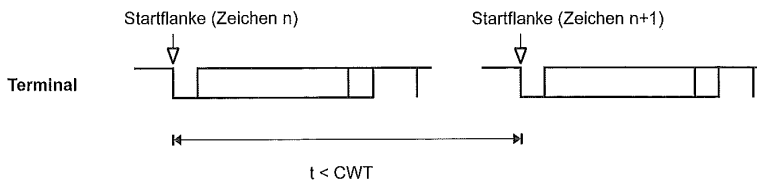


Bild 6.33 Die Definition der Zeichenwartzeit CWT.

Die CWT berechnet sich nach folgender Formel aus dem im ATR enthaltenen Datenelement CWI.

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

Der voreingestellte Wert für CWI ist 13, sodass sich die folgende CWT ergibt:

$$CWT = (2^{13} + 11) \text{ work etu} = 8\,203 \text{ work etu}$$

Bei einem Takt von 3,5712 MHz und einem Teiler von 372 errechnet sich daraus eine Zeitdauer von 0,85 Sekunden.¹⁴

Diese Zeitdauer, die als Grundeinstellung in der Norm vorgegeben ist, ist für eine schnelle Datenübertragung zu hoch. Der in der Praxis übliche Bereich für CWI liegt zwischen 3 und 5. Dies bedeutet bei einer normalen Sendefolge, bei der ein Zeichen nach dem Anderen ohne zeitliche Verzögerung über die Schnittstelle geht, dass der Empfänger noch die Zeitspanne von ein bis zwei Byte wartet, bis er das Ende des Blocks oder einen Kommunikationsabbruch erkennt.

Normalerweise erkennt die Empfangsroutine das Ende eines Blocks anhand der Blocklängenangabe im Längenfeld LEN. Ist der Inhalt dieses Feldes jedoch fehlerhaft, nutzt man zusätzlich die Zeichenwartzeit, um einen definierten Abbruch beim Empfang zu erreichen. Dieses Problem wird aber auch nur dann akut, wenn die Längenangabe zu lang ist, denn in diesem Fall würde der Empfänger auf weitere Zeichen warten, die nie eintreffen. Damit wäre das Übertragungsprotokoll blockiert, und dieser Zustand könnte nur mehr durch einen Reset der Chipkarte beseitigt werden. Durch den Mechanismus der Zeichenwartzeit umgeht man dies.

Blockwartzeit BWT (*block waiting time*)

Um einen definierten Abbruch der Kommunikation mit einer nicht antwortenden Chipkarte zu erreichen, wurde die Blockwartzeit BWT definiert. Es ist die maximale Zeitspanne zwischen der Startflanke des letzten Bytes eines Blocks, der zur Karte gesendet wurde, und der Startflanke des ersten Bytes, das die Karte zurücksendet.

¹⁴ Siehe auch Abschnitt 16.11.2 „Umrechnungstabelle für Datenelemente des ATR“.

Auf einen üblichen $T = 1$ Block übertragen, ist dies die maximale Zeit, die zwischen der Startflanke des XOR-Bytes im Epilogfeld im Kommandoblock und der Startflanke des NAD-Bytes in der Antwort der Karte vergehen darf.

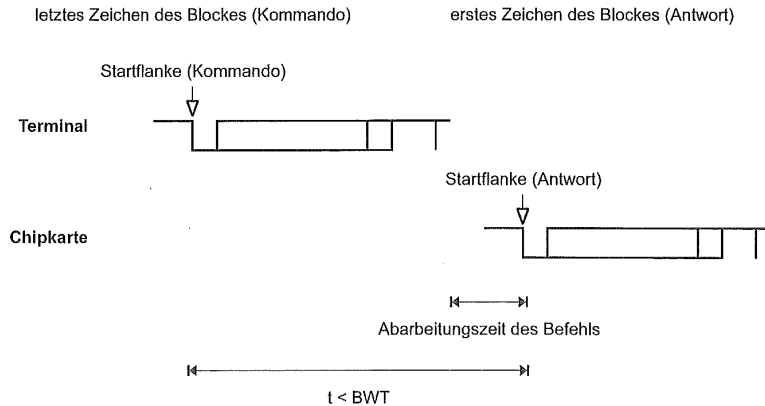


Bild 6.34 Die Definition der Blockwartezeit BWT.

Läuft diese Zeit ab, ohne dass die Karte eine Antwort sendet, darf das Terminal von einer Fehlfunktion der Karte ausgehen und entsprechende Reaktionsmechanismen einleiten. Dies wäre etwa ein Reset der Karte mit dem anschließenden Versuch des Neuaufbaus der Kommunikation.

Im ATR wird in den Interface Characters ein Wert für die BWT angegeben, der verkürzt als BWI codiert ist.

$$BWT = 2^{BWI} \cdot 960 \cdot \frac{372}{f} \cdot s + 11 \text{ work etu}$$

Ist im ATR kein Wert für BWI definiert, so benutzt man die Grundeinstellung von $BWI = 4$. Damit ergibt sich bei 3,5712 MHz und Teiler 372 der Wert 1,6 s für die Blockwartezeit.

$$BWT = 2^4 \cdot 960 \cdot \frac{372}{3\,571\,200 \text{ Hz}} \cdot s + 11 \text{ work etu} = 2^4 \cdot 0,1 \cdot s + 11 \text{ work etu} \approx 1,6 \text{ s}$$

Wie man sieht, ist dieser Wert großzügig bemessen. In der Praxis benutzt man für BWI oft den Wert 3, der zu einer Blockwartezeit von 0,8 s führt.¹⁵ Die typischen Ausführungszeiten von Kommandos in der Chipkarte liegen in der Regel im Bereich von 0,2 s¹⁶. Deshalb ist eine BWT der vorstehend angegebenen Dauer ein Kompromiss zwischen üblicher Ausführungsdauer von Kommandos und der schnellen Detektion einer nicht mehr auf Kommandos antwortenden Chipkarte.

¹⁵ Siehe auch Abschnitt 16.11.2 „Umrechnungstabelle für Datenelemente des ATR“.

¹⁶ Siehe auch Abschnitt 15.2 „Formelsammlung zur rechnerischen Abschätzung von Ausführungszeiten“.

Blockschutzzeit BGT (*block guardtime*)

Die minimale Zeitspanne zwischen der Startflanke des letzten Bytes und der Startflanke des ersten Bytes in der entgegengesetzten Übertragungsrichtung ist definiert als die „Blockschutzzeit BGT“. Sie ist das Gegenteil der BWT, die ja als die maximale Zeitspanne zwischen den beiden definierten Startflanken festgesetzt ist. Ein weiterer Unterschied besteht darin, dass die Blockschutzzeit für beide Übertragungspartner verbindlich ist und eingehalten werden muss, die Blockwartezeit aber nur für die Chipkarte von Bedeutung ist. Zweck der Blockschutzzeit ist es, dem Sender eines Blocks eine Mindestzeitspanne einzuräumen, in der er von Senden auf Empfangen umschalten kann.

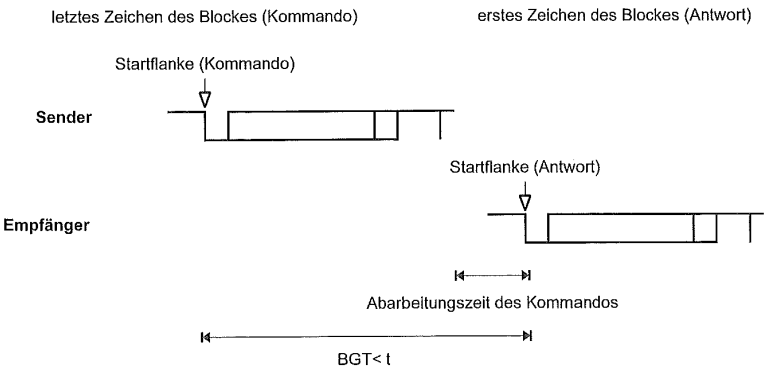


Bild 6.35 Die Definition der Blockschutzzeit BGT.

Der Wert der Blockschutzzeit ist ein Fixum und beträgt nach Norm 22 etu. Für eine mit 3,5712 MHz betriebene Chipkarte mit Teiler 372 ergibt sich somit eine Zeitspanne von ca. 2,3 ms.

6.4.3.4 Mechanismen des Übertragungsprotokolls

Wartezeitverlängerung (*waiting time extension*)

Benötigt eine Karte für die Erzeugung einer Antwort mehr Zeit, als durch die Blockwartezeit BWT maximal erlaubt ist, kann sie eine Wartezeitverlängerung vom Terminal anfordern. Dazu sendet die Chipkarte einen speziellen S-Block, der eine Wartezeitverlängerung anfragt, und erhält darauf vom Terminal einen passenden S-Block als Bestätigung dieser Anfrage. Eine Ablehnung dieser Anfrage durch das Terminal ist nicht erlaubt.

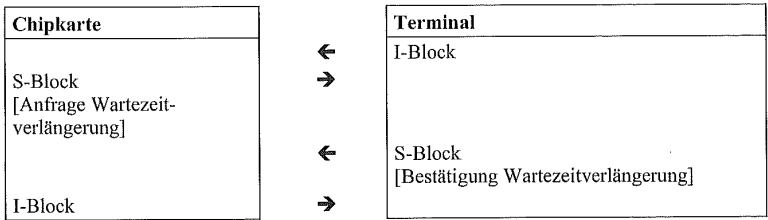


Bild 6.36 Der Ablauf der Wartezeitverlängerung

Die Dauer der Verlängerung der Blockwartzeit wird dem Terminal durch ein Byte im Informationsfeld mitgeteilt. Dieses Byte mit der Blockwartzeit multipliziert ergibt die neue Blockwartzeit, die allerdings nur für die Abarbeitung des zuletzt gesendeten I-Blocks gültig ist.

Blockverkettung (*chaining*)

Eines der wesentlichen Leistungsmerkmale des T=1 Protokolls ist die Funktion der Blockverkettung. Sie erlaubt jeweils einem der beiden Kommunikationspartner, Datenblöcke zu übertragen, die größer als der betreffende Sende- bzw. Empfangspuffer sind. Dies kann gerade im Bereich des begrenzten Speicherplatzes der Chipkarten sehr nützlich sein. Erlaubt ist diese Verkettungsfunktion nur bei Informationsblöcken, da diese als einziger Blocktyp große Datenlängen aufweisen können. Die Anwendungsdaten werden bei der Blockverkettung in Einzelblöcke aufgeteilt und nacheinander zum Empfänger geschickt.

Die Daten für die Anwendungsschicht müssen so aufgeteilt werden, dass keines der dabei entstehenden Teile größer als die maximale Blockgröße des Empfängers ist. Danach wird der erste Teil im Rahmen des T=1 Protokolls in ein Informationsfeld eingebettet, mit Prolog- und Epilogfeld versehen und zum Empfänger gesendet. Im PCB-Feld des Blocks ist dabei das M-Bit (*more data bit*) gesetzt. Dieses zeigt dem Empfänger an, dass nun die Blockverkettungsfunktion Verwendung findet und in den nachfolgenden Blöcken verkettete Daten folgen.

Sobald der Empfänger diesen Informationsblock mit dem ersten Teil der Anwendungsdaten korrekt empfangen hat, signalisiert er mit einem R-Block, dessen Folgenummer $N(R)$ der Sendefolgezähler $N(S)$ des nächsten I-Blocks ist, dass er bereit ist, den nächsten verketteten I-Block zu empfangen. Daraufhin erhält der Empfänger vom Sender diesen Block.

Dieses Hin und Her von I- und R-Blöcken setzt sich so lange fort, bis vom Sender ein I-Block ausgeht, dessen M-Bit im PCB-Feld anzeigt, dass dies der letzte Block der Kette ist ($M\text{-Bit} = 0$). Nach Empfang dieses Blocks befinden sich alle Daten der Anwendungsschicht beim Empfänger, und dieser kann nun den Block verarbeiten.

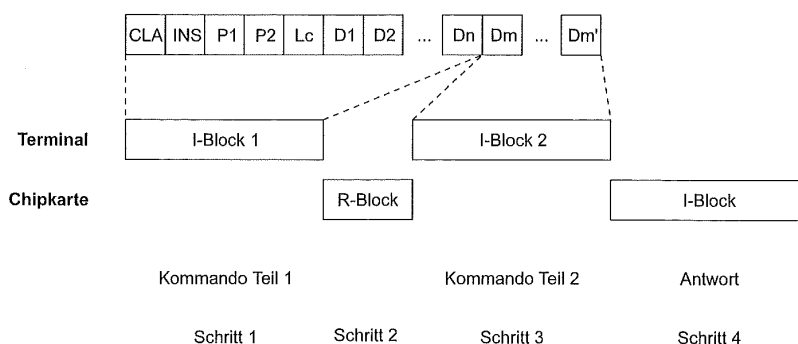


Bild 6.37 Beispiel der Blockverkettung bei der Übertragung eines Informationsblocks vom Terminal zur Chipkarte.

Für den Ablauf von Blockverkettungen existiert allerdings eine Einschränkung. Innerhalb eines Kommando-Antwort-Zyklus darf eine Verkettung nur in eine Richtung vorgenommen werden. Wäre also im oben erläuterten Beispiel das Terminal der Sender der verketteten Blöcke, dürfte die Karte als Antwort darauf keine verketteten Blöcke senden.

Es gibt noch eine weitere Einschränkung, die allerdings nicht wie die vorausgehende protokolltechnischer Natur, sondern durch den sehr begrenzten Speicherplatz der Chipkarten verursacht ist. Die Implementation des Mechanismus der Blockverkettung in Software ist mit einigem Aufwand verbunden, der Nutzen jedoch stark beschränkt, da die Kommandos und Antworten keine allzu große Länge haben und deshalb normalerweise keine Blockverkettung benötigen.

Falls die Länge des in der Chipkarte im RAM zur Verfügung stehenden Empfangspuffers nicht ausreicht, um dort alle Daten bei einer Blockverkettung zu speichern, ist es notwendig, diesen Puffer im EEPROM anzulegen. Dies bedingt allerdings eine starke Reduzierung der Übertragungsgeschwindigkeit, da der EEPROM-Schreibzugriff im Gegensatz zum RAM nicht mit voller Prozessorgeschwindigkeit durchgeführt werden kann.

Deshalb haben manche $T = 1$ Implementierungen keine Funktion zur Blockverkettung, da der dafür notwendige Aufwand im Verhältnis zum Nutzen oft nicht gerechtfertigt ist. Dies ist auch ein typisches Beispiel dafür, dass in der Praxis Normen oft sehr großzügig interpretiert werden. In diesem Fall läuft die Interpretation dahingehend, dass eine Blockverkettung bei $T = 1$ eine zusätzliche Option ist, die nicht unbedingt notwendig ist. Dies kann jedoch in der Praxis zu erheblichen Kompatibilitätsproblemen führen.

Fehlerbehandlung

Das $T = 1$ Protokoll weist hoch entwickelte Fehlererkennungs- und Behandlungsmechanismen auf. Werden ungültige Blöcke empfangen, so versucht das Protokoll anhand genau festgelegter Abläufe die Kommunikation wieder auf ein fehlerfreies Niveau zurückzuführen.

Dabei gibt es aus der Sicht des Terminals drei Synchronisationsstufen. In der ersten Stufe erhält der Sender eines fehlerhaften Blocks einen R-Block, der einen EDC-/Paritätsbitfehler oder einen allgemeinen Fehler anzeigt. Der Empfänger dieses R-Blocks muss daraufhin den letzten gesendeten Block wiederholen.

Ist es nicht möglich, mit dem Mechanismus der ersten Stufe eine fehlerfreie Verbindung wiederherzustellen, geht man zur zweiten Stufe über. Dies bedeutet, dass die Chipkarte vom Terminal eine Resynchronisations-Anfrage in einem S-Block erhält. Das Terminal erwartet daraufhin eine Resynch-Antwort. Gleichzeitig setzt sowohl das Terminal als auch die Chipkarte die Sende- und Empfangszähler auf null zurück. Dies entspricht dem Protokollzustand direkt nach dem ATR. Auf der Grundlage dieses Ursprungszustandes versucht nun das Terminal, eine neue Verbindung aufzubauen.

Stufe 1 und 2 beeinflussen nur die Protokollschicht. Sie haben keinerlei Auswirkungen auf die Anwendung selber. Die dritte Stufe der Synchronisation allerdings betrifft alle Schichten in der Chipkarte. Kann das Terminal durch die Synchronisationsstufen 1 und 2 keine fehlerfreie Verbindung mehr aufbauen, löst es mittels der Reset-Leitung zur Chipkarte einen Reset aus. Allerdings gehen dabei alle Informationen und Zustände der aktuellen Sitzung verloren. Im Anschluss an diesen Reset muss die Kommunikation komplett neu aufgebaut werden.

Führt auch dieses Verfahren zu keiner funktionierenden Verbindung, so deaktiviert das Terminal nach dreimaligem Versuch die Chipkarte. Üblicherweise erhält der Benutzer danach eine Fehlermeldung, dass seine Karte kaputt ist.

Tabelle 6.39 Die Stufen der Fehlerbehandlung bei $T = 1$

Synchronisationsstufe	Mechanismus
Stufe 1	Wiederholung des fehlerhaften Blocks
Stufe 2	Resynchronisation und Wiederholung des fehlerhaften Blocks
Stufe 3	Reset der Chipkarte und Neuaufbau der Verbindung

6.4.3.4 Beispiel für die Datenübertragung mit $T = 1$

Das folgende Bild zeigt beispielhaft die Übertragung des Kommandos SELECT FILE mit $T = 1$.

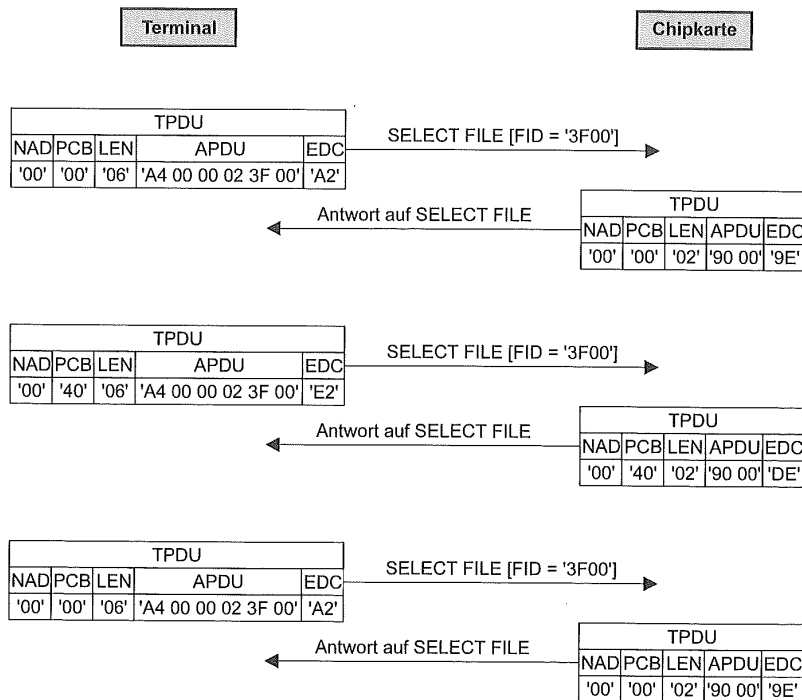


Bild 6.38 Der Gutfall der Datenübertragung einer TPDU mit dem Übertragungsprotokoll $T = 1$. Als Fehlererkennungscode (EDC) wurde die Option XOR gewählt. In der APDU wird das Kommando SELECT FILE mit dem FID '3F00', d. h. Selektion des MFs, übertragen. Man sieht sehr gut, wie im PCB-Byte der Sendefolgezähler bei jedem Sendevorgang erhöht wird und sich der EDC entsprechend ändert.

6.4.3.5 Unterschiede zwischen T = 1 nach ISO/IEC und T = 1 nach EMV

In der ursprünglichen Festlegung des T = 1 Protokolls nach ISO/IEC 7816-3 sind viele Mechanismen und Optionen vorgesehen, die zum Teil programmcodeaufwendig sind und auch oft nicht genutzt werden. Das typische Beispiel sind die umfangreichen Fehlerbehebungsmechanismen, welche zwar theoretisch interessant sind, aber bei einer echten Übertragungsstörung oftmals nicht greifen. Es ist in der Praxis meist besser, die Chipkarte nochmals in das Terminal zu stecken und die Sitzung von Anfang an neu aufzusetzen, als mit zahlreichen Resynchronisationsanfragen zu versuchen, die Übertragung wieder zu stabilisieren. Deshalb wurden im EMV-Standard einige Einschränkungen gegenüber der ursprünglichen ISO/IEC-Norm getroffen.

Tabelle 6.40 Überblick der Unterschiede bei Implementationen des Übertragungsprotokolls T = 1 nach ISO/IEC 7816-3 bzw. nach EMV

Mechanismus/Option	ISO/IEC 7816-3	EMV
BWT abgelaufen	z. B.: Reset der Karte	Deaktivierung der Karte
Chipkarte sendet Anfrage zur IFS Änderung	erlaubt	maximal 3-mal nacheinander erlaubt
Chipkarte sendet S-Block mit Abort Request	erlaubt	Deaktivierung der Karte
I-Block mit Länge null	erlaubt	verboten
Terminal sendet nacheinander 3 Blöcke und erhält keine gültige Antwort	Verhalten analog festgelegter Fehlerbehandlung, üblicherweise Resynch Anfrage.	Deaktivierung der Karte

6.4.4 Übertragungsprotokoll T = 14 (Deutschland)

Die ISO-Norm 7816-3 sieht im ATR die Kennzeichnung eines nationalen Übertragungsprotokolls vor. Die reservierte Bezeichnung dafür lautet T = 14. Mit der Einführung des C-Netzes für Mobiltelefone und der Kartentelefone in Deutschland benötigte man ein Protokoll zur Kommunikation mit den eingesetzten Chipkarten. Das zeichenorientierte Protokoll T = 0 wollte man nicht verwenden und ein genormtes Blockprotokoll gab es noch nicht. Deshalb entschied sich die Telekom 1987, einen von einem DIN-Arbeitskreis erstellten Normenvorschlag zu verwenden. Dieses Protokoll erhielt die Bezeichnung T = 14, was nichts anderes bedeutet, als dass es eine eigene nationale Lösung ist. Außerhalb von Deutschland hatte dieses Protokoll keine Bedeutung, es hatte allerdings sehr großen Einfluss auf die spätere Entwicklung des international genormten Protokolls T = 1, da T = 14 dafür die wichtigste Grundlage war.

Der Verbreitungsgrad von T = 14 in Deutschland war durch das Mobilfunknetz C-Netz und die öffentlichen Kartentelefone sehr groß. Mit dem Betriebsende des C-Netzes Ende 2000 und der Umstellung der öffentlichen Kartentelefone auf das T = 1 Protokoll hat T = 14 in Deutschland seine Bedeutung verloren, weshalb es hier nicht im Detail beschrieben ist.

Das Übertragungsprotokoll T = 14 ist blockorientiert aufgebaut und arbeitet asynchron zum angelegten Takt. Der dabei verwendete Teiler (*clock rate conversion factor*) beträgt

512, sodass bei einer Taktfrequenz von 4,9152 MHz eine Übertragungsgeschwindigkeit von 9600 Bit/sec zustande kommt. Die Datenübertragung auf Schicht 2 (Leitungsschicht) wird immer in direct convention durchgeführt. Die Größe des Puffers für Übertragungsblöcke muss mindestens 50 Byte betragen, der maximale Wert beträgt 255 Byte. Ein Mechanismus zur Blockverkettung (*chaining*) ist nicht vorgesehen.

6.4.5 Übertragungsprotokoll USB

In einem Anhang zur ISO/IEC 7816-3 wird sich in Zukunft die Spezifikation eines neuen Übertragungsprotokolls für Chipkarten befinden. Es ist dies das USB (*universal serial bus*)-Protokoll, das sich gegenüber den Konkurrenten wie Firewire und Ähnlichen bei Chipkarten durchgesetzt hat.

USB erfordert auf dem Chipkarten-Mikrocontroller eine spezielle Hardware-Komponente, die sich jedoch bereits auf vielen Chips zumindest optional befindet. Der große Vorteil von USB gegenüber den herkömmlichen Übertragungsprotokollen von Chipkarten besteht darin, dass es sich um einen etablierten Standard aus dem PC-Umfeld handelt. Zusätzlich bietet USB eine höhere Übertragungsrate als $T = 0$ oder $T = 1$.

Bei Chipkarten wird voraussichtlich der USB-Standard in der Version 1.1 zum Einsatz kommen. Dabei wird je nach Mikrocontroller sowohl die Low-speed-Variante mit 1,5 MBit/s als auch Full-speed-Option mit 12 MBit/s unterstützt. Die effektive Übertragungsrate nach Abzug der für die Übertragung notwendigen Protokolldaten liegt allerdings deutlich unten den angegebenen Werten. USB Version 2.0 mit bis zu 480 MBit/s (High-speed-Variante) wird in absehbarer Zeit nicht zur Anwendung kommen.

Für die Realisierung von USB auf Chipkarten werden die seit langer Zeit festgelegten aber bisher nicht genutzten Kontakte C4 (AUX1) und C8 (AUX2) zum Einsatz kommen.

Weitere Details werden in Zukunft in der vorgenannten Norm festgelegt, doch ist absehbar, dass es noch einige Jahre dauern wird, bis alle diesbezüglichen Normierungsarbeiten abgeschlossen sind.

6.4.6 Vergleich der asynchronen Übertragungsprotokolle

Eine kurze Anmerkung zur erzielbaren Übertragungsgeschwindigkeit sei hier und in diesem Zusammenhang noch angebracht. Oft versucht man die beiden Übertragungsprotokolle $T = 0$ und $T = 1$ zu vergleichen, indem man eine effektive Übertragungsrate berechnet. Dies gilt dann aber nur für ein bestimmtes Kommando und in einem bestimmten Zusammenhang. Sobald man aber solche Berechnungen verallgemeinert, verlieren sie ihre Bedeutung und auch ihre Richtigkeit.

Beide Protokolle haben ihre Stärken und Schwächen bei der erreichbaren Übertragungsgeschwindigkeit, die aber von sehr vielen einzelnen Faktoren abhängen. Dies kann beispielsweise die Fehlerhäufigkeit bei der Übertragung, Größe des I/O-Puffers in der Chipkarte oder die spezielle Implementation des Protokolls sein. Zusammenfassend kann man davon ausgehen, dass beide Protokolle im Durchschnitt bei den meisten Anwendungen annähernd die gleiche effektive Übertragungsrate haben. Will man diese erhöhen, wird ein Wechsel des Protokolls wenig bewirken. Besser ist es, den Teiler zu erniedrigen, da man mit diesem Ansatzpunkt wesentlich mehr erreichen kann.

In den vorangehenden Abschnitten wurden zwei internationale Übertragungsprotokolle beschrieben. Um den Überblick zu wahren, sind in der Tabelle 6.41 die wesentlichen Eigenschaften sowie Vor- und Nachteile in kurzer Form zusammengefasst.

Tabelle 6.41 Die international genormten asynchronen Übertragungsprotokolle im Vergleich

Kriterium	T = 0	T = 1	T = 2 (Normungsentwurf)
Datenübertragung	asynchron, halbduplex, byteorientiert	asynchron, halbduplex, blockorientiert	asynchron, voll duplex, blockorientiert
Norm	ISO/IEC 7816-3, GSM 11.11, EMV	ISO/IEC 7816-3, EMV	ISO/IEC 10 536-4
Teiler	frei definierbar, üblich: 372	frei definierbar, üblich: 372	frei definierbar, üblich: 372
Blockverkettung	nicht möglich	möglich	möglich
Fehlererkennung	Paritätsbits	Paritätsbits, EDC am Blockende	Paritätsbits, EDC am Blockende
Speicherbedarf für Implementierung	ca. 300 Byte	ca. 1 100 Byte	ca. 1 600 Byte

6.5 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 (*transport 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 sozusagen 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.

6.5.1 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.

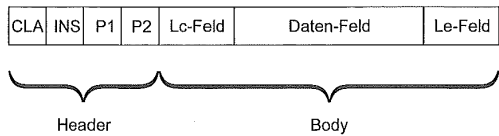


Bild 6.39 Die Struktur einer Kommando-APDU.

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 Codierung '8X' benutzt. Kommandos nach ISO sind hingegen mit dem Class Byte '0X' codiert. 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.

Tabelle 6.42 Die wichtigsten Codierungen 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 Codierung nach ISO/IEC 7816-4/-7/-8
'8', '9'	...	...	...	...	Struktur nach ISO/IEC 7816-4, anwendereigene Codierung und Bedeutung der Kommandos und Antworten (<i>private use</i>)
'A'	...	...	...	...	Struktur und Codierung nach ISO/IEC 7816-4, spezifiziert in zusätzlichen Dokumenten (z.B. GSM 11.11)
'F'	1	1	1	1	Reserviert für PPS

Tabelle 6.43 Überblick über die Zuordnung von Class Bytes zu 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 (<i>private use</i>)
'8X'	für Kreditkarten mit Chip nach EMV
'A0'	für GSM-Mobilfunksystem GSM 11.11

Das nächste Byte in der Kommando-APDU ist das Instruction-Byte, mit dem die eigentlichen Kommandos codiert werden. Dabei kann fast der gesamte Adressraum dieses Bytes mit der Einschränkung nur geradzahlig Codierungen ausgeschöpft werden. Dies ist notwendig, weil das T = 0 Protokoll die Programmierspannung durch ein um eins erhöhtes Instruction-Byte im Procedure-Byte aktivieren kann. Somit muss zur Unterscheidung das Instruction-Byte immer geradzahlig sein.¹⁷

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 zur Auswahl der verschiedenen Selektionsmöglichkeiten bei SELECT FILE oder zur Angabe des Offsets bei READ BINARY.

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 (Lc-Feld¹⁸) und spezifiziert auch die Länge des von der Karte zurück zu sendenden Datenteils (Le-Feld).¹⁹ Zum Zweiten enthält er die zum Kommando gehörenden Daten, die zur Karte gesendet werden. Falls das Le-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 Le- und Lc-Feld immer eine Länge von einem Byte. Es ist möglich, daraus ein Le-/Lc-Feld mit je drei Byte Länge zu machen. Damit könnten Längen bis 65 536 dargestellt werden, da das erste Byte als Escape-Sequenz '00' codiert wäre. Die Norm definiert diese Längenangabe mit drei Byte für zukünftige Anwendungen. Aufgrund der heutigen Speichergrößen gibt es noch keine Realisationen.

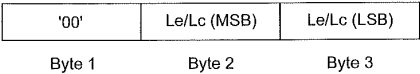


Bild 6.40 Die Struktur eines erweiterten Lc-/Le-Feldes.

Durch die Kombination der vorgehend beschriebenen Teile der Kommando-APDU kann man die folgenden vier generellen Fälle, so genannte cases, unterscheiden:

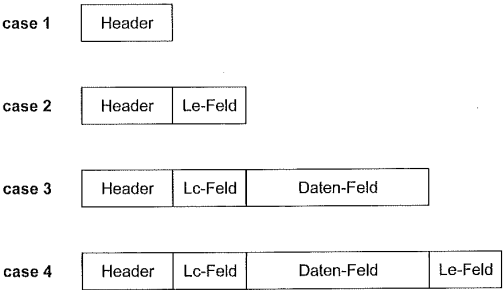


Bild 6.41 Die vier möglichen cases von Kommando-APDUs.

¹⁷ Siehe auch Abschnitt 16.11.5 „Tabelle der wichtigsten Chipkarten-Kommandos“.

¹⁸ Lc: *length command*

¹⁹ Le: *length expected*

6.5.2 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.

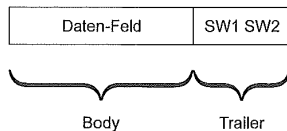


Bild 6.42 Die Struktur einer Antwort-APDU.

Der Body besteht aus dem Datenfeld, dessen Länge in der vorangehenden Kommando-APDU mit dem Le-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 ein Byte langen Status-Wörtern 1 und 2 im Trailer angezeigt.

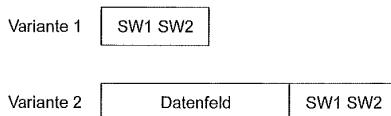


Bild 6.43 Die zwei Varianten der Antwort-APDUs.

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 Codes zugrunde liegt, ist in Bild 6.51 dargestellt.²⁰

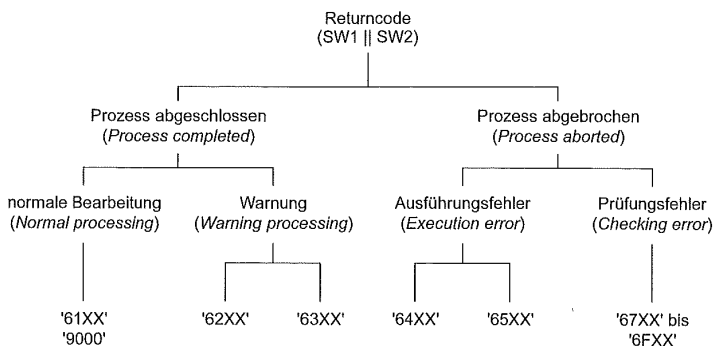


Bild 6.44 Die Systematik der Returncodes nach ISO/IEC 7816-4. Bei den Returncodes '63XX' und '65XX' wurden Daten im nichtflüchtigen Speicher (d. h. EEPROM) verändert, bei den restlichen Returncodes mit '6X' nicht.

²⁰ Siehe auch Abschnitt 16.11.8 „Chipkarten-Returncodes“.

Erhält man nach der Ausführung eines Kommandos den Returncode '63XX' oder '65XX', dann wurde der Zustand des nichtflüchtigen Speichers (d. h. in der Regel das EEPROM) in der Chipkarte geändert. Empfängt man hingegen einen der restlichen Returncodes, die mit '6X' beginnen, dann 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 vielen Anwendungen davon abweichende Codierungen benutzt werden. Die einzige Ausnahme stellt der Code '9000' dar, der praktisch überall den Gutfall symbolisiert. Bei allen anderen Codes sollte man immer in der zugrunde liegenden Spezifikation nachlesen, um die Bedeutung zweifelsfrei herauszufinden.

6.6 Sicherung der Datenübertragung

Der gesamte Datenaustausch zwischen Terminal und Chipkarte findet durch digitale elektrische Impulse über die I/O-Leitung der Chipkarte statt. Es ist vorstellbar und auch technisch einfach realisierbar, 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 aufwendiger 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 eigene Kommandos einzuspielen.

Diese beiden typischen Angriffe würden nur dann zu einem 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 aufwendigere Angriffe abzuwehren, gibt es verschiedene Mechanismen und Verfahren. Diese bezeichnet man allgemein als Secure Messaging. 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.

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 noch erfüllt sein: Alle Sicherheitsmechanismen müssen sich gegenüber den vorhandenen Protokollschichten völlig transparent verhalten, sodass 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 Kerckhoff, 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 vielen Jahren existieren verschiedene Variationen von Verfahren für Secure Messaging. Allen war gemeinsam, dass sie relativ starr und für eine bestimmte Anwendung maßgeschneidert waren. Vom Standpunkt der Sicherheit sind bei den meisten keine Einwendungen zu machen. Allerdings setzte sich keines dieser Verfahren international so weit durch oder war so flexibel, dass es Eingang in die aktuelle Normung fand.

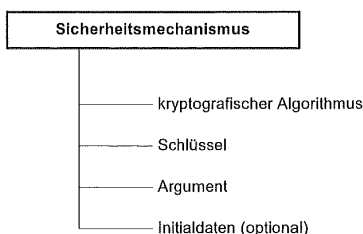


Bild 6.45 Notwendige Daten und Funktionen für einen Sicherheitsmechanismus.

Die Forderungen nach Transparenz zu vorhandenen Kommandos, zwei grundverschiedenen Übertragungsprotokollen und einer möglichst hohen Variabilität hatte zur Folge, dass ein sehr flexibles, aber dadurch auch aufwendiges und komplexes Verfahren für Secure Messaging in der ISO/IEC 7816-4 und erweiterte Funktionen dazu in der ISO/IEC 7816-8 genormt wurde.²¹ Die Grundlage stellt dabei die Einbettung aller Nutzdaten in TLV-codierte 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 codiert werden, ob das Verfahren nach ISO/IEC 7816-4/8 verwendet wird und ob der Header in die kryptografische Prüfsumme (CCS) mit eingeschlossen ist.²² Ist der Header bei der Berechnung mit eingeschlossen, so ist er authentisch, da er während der Übertragung nicht unbemerkt verändert werden kann.

²¹ Secure Messaging wird üblicherweise als „SM“ abgekürzt, was aber aufgrund der vielen Freiheitsgrade und Interpretationsspielräume der beiden Normen viele Entwickler oft auch als „sado/maso“ übersetzen.

²² Codierung des Class Byte, siehe Abschnitt 6.5.1 „Struktur der Kommando-APDUs“.

Datenobjekte für Klartext

Nach der Norm müssen alle Daten, die nicht BER-TLV-codiert sind, in Datenobjekte eingebettet, d. h. gekapselt werden. Dazu gibt es verschiedene Kennzeichen, die in der folgenden Tabelle 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 6.44 Die Kennzeichen für Klartext-Datenobjekte

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

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 beiden folgenden Tabellen.

Tabelle 6.45 Die Kennzeichen für Datenobjekte für Vertraulichkeit

Kennzeichen	Bedeutung
'82', '83'	Kennzeichen für ein Kryptogramm: der Klartext ist BER-TLV-codiert und schließt Datenobjekte für Secure Messaging ein
'84', '85'	Kennzeichen für ein Kryptogramm: der Klartext ist BER-TLV-codiert 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 6.46 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.

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 aufwendigere 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, sodass 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, was die Bezeichnung „diversity“ zum Ausdruck bringt.

6.6.1 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 vom Empfänger bemerkt wird.

Die Benutzung des Verfahrens wird dabei durch ein Bit im Class-Byte angezeigt, sodass 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 durch den Angreifer empfangen und ausgewertet werden. Dies muss nicht unbedingt ein 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 sehen, welche Daten von seiner Chipkarte und dem Terminal ausgetauscht werden.

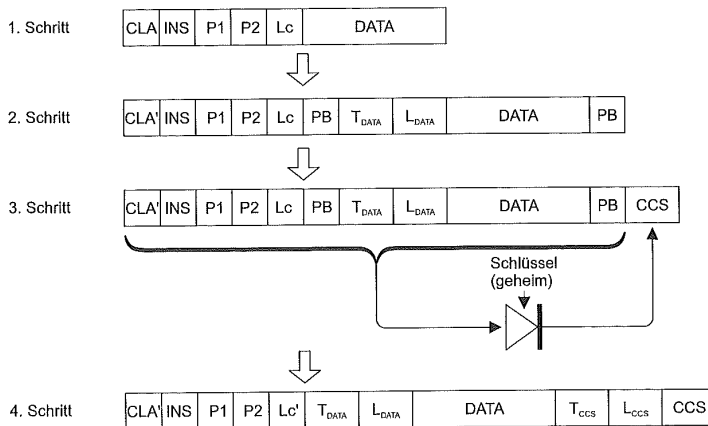


Bild 6.46 Erstellung einer Kommando-APDU im Authentic-Verfahren. Es wird ein Case-3-Kommando (z. B. UPDATE BINARY) verwendet, dessen Header in die kryptografische Prüfsumme (CCS) mit einbezogen ist. Die Erstellung einer Antwort-APDU verläuft analog. Die Padding Bytes sind als PB abgekürzt.

1. Schritt: Ausgangsformat der APDU.
2. Schritt: Transformation des Datenteils in TLV-codierte Daten und Padding der Datenobjekte auf ein Vielfaches von acht.
3. Schritt: Berechnung der CCS.
4. Schritt: Ergänzung der APDU mit einem TLV-codierten Datenobjekt, das die CCS enthält.

Zur Berechnung der kryptografischen Prüfsumme kann im Prinzip jeder Blockverschlüsselungsalgorithmus verwendet werden. In den folgenden Ausführungen wurde dabei aus praktischen Gründen immer vom DES 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 DES-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-codiertes 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 findet. 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 DES-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 dazu üblicherweise dynamische Schlüssel. Diese werden durch Verschlüsselung einer zwischen Terminal und Chipkarte vorab ausgetauschten Zufallszahl mit einem beiden Kommunikationsteilnehmern bekannten gemeinsamen Schlüssel erzeugt.

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

6.6.2 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.

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 DES im CBC-Mode verschlüsselt. Ausgespart bleibt dabei der Header, was aus Kompatibilitätsgründen zum T = 0 Protokoll notwendig ist. Möchte man den Header ebenfalls verschlüsseln, sodass 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.

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 DES miteinander verknüpft sind. Damit würde vom Empfänger ein Austausch sofort bemerkt.

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

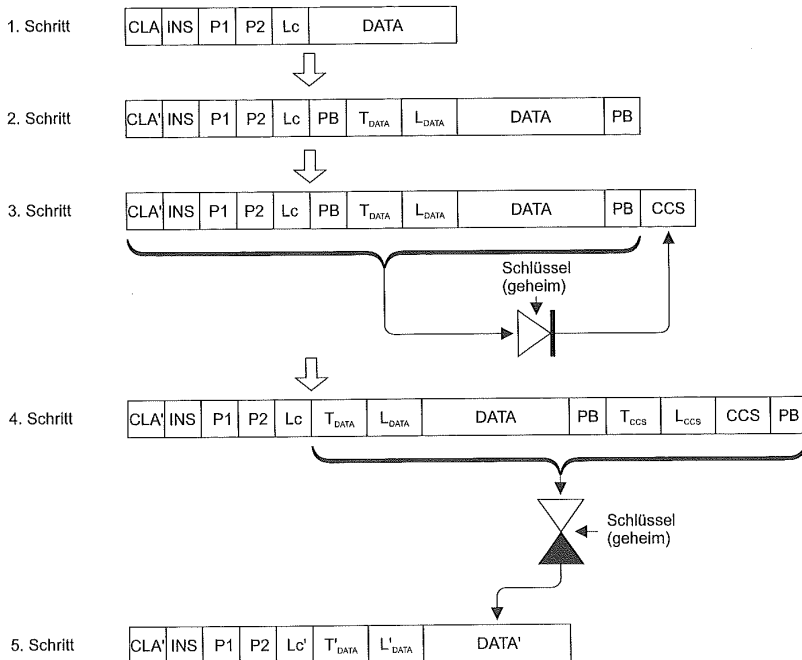


Bild 6.47 Erstellung einer Kommando-APDU im Combined-Verfahren. Es wird ein Case-3-Kommando (z. B. UPDATE BINARY) verwendet, dessen Header in die kryptografische Prüfsumme (CCS) mit einbezogen ist. Die Erstellung einer Antwort-APDU läuft analog. (Padding Bytes - PB)

1. Schritt: Ausgangsformat der APDU.
2. Schritt: Transformation des Datenteils in TLV-codierte Daten und Padding der Datenobjekte auf ein Vielfaches von acht.
3. Schritt: Berechnung der CCS.
4. Schritt: Ergänzung der APDU mit einem TLV-codierten Datenobjekt für die CCS, Padding der Datenobjekte auf ein Vielfaches von acht.
5. Schritt: Verschlüsselung des Datenteils der APDU.

temporärer Natur sein, sodass für jede Sitzung ein eigens abgeleiteter Schlüssel Verwendung findet. 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 erhebliche Senkung der Datenübertragungsgeschwindigkeit verbunden.

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 vier angenommen werden. Der zeitliche Unterschied zwischen Authentic- und Combined-Verfahren beläuft sich damit immerhin noch auf den Faktor zwei. Es ist deshalb in jedem Einzelfall sehr genau zu prüfen, welche Daten auf solch aufwendige, aber sichere Weise übertragen werden sollen.

6.6.3 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 (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.

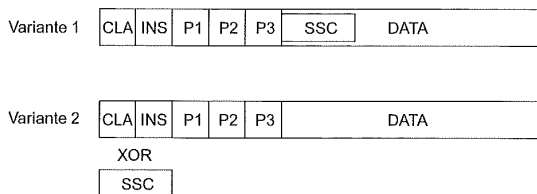


Bild 6.48 Die beiden Varianten eines Sendefolgezählers innerhalb einer Kommando-APDU. Variante 1 zeigt einen Sendefolgezähler SSC (*send sequence counter*) als TLV-codiertes Datenobjekt im Datenteil, wogegen bei Variante 2 ein Sendefolgezähler nur für die CCS-Berechnung durch XOR mit den APDU-Daten verknüpft ist.

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.

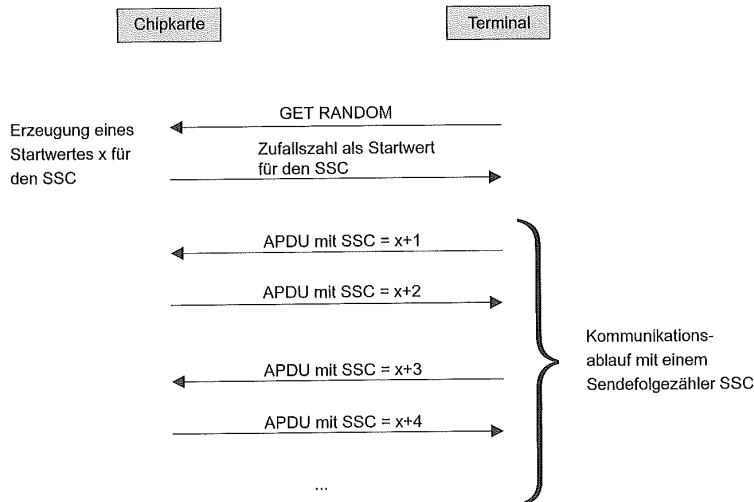


Bild 6.49 Übertragung von APDUs mit einem Sendefolgezähler SSC (*send sequence counter*).

6.7 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 vier Anwendungen auf einer Chipkarte parallel mit einem Terminal Daten austauschen können. Benutzt wird nach wie vor die nur einmal vorhandene serielle Schnittstelle, doch auf logischer Ebene können die Anwendungen einzeln adressiert werden.

Die Unterscheidung, welches Kommando nun zu welcher Anwendung gehört, wird durch zwei Bits im Class Byte vorgenommen (Bit 1 und Bit 2).²³ Damit sind bis zu vier logische Kanäle verfügbar, und es können bis zu vier Sitzungen parallel mit Anwendungen

²³ Siehe auch Abschnitt 6.5.1 „Struktur der Kommando-APDUs“.

in der Chipkarte ablaufen.²⁴ 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, denn die Antwort-APDU der Karte enthält keine Angabe des ursprünglichen logischen Kanals. Es kann also von außen nicht festgestellt werden, welche Antwort welchen Returncode zurückgesendet hat. Das Resultat dieser fehlenden Kanalangebe ist, dass nach jedem Kommando an die Karte auf die daran anschließende Antwort gewartet werden muss.

Das Einsatzgebiet dieses sehr mächtigen Mechanismus ist vor allem die gleichzeitige Benutzung von mehreren Anwendungen. Beispielsweise führt ein Benutzer ein Telefongespräch, wozu er die GSM-Anwendung in der Multiapplikations-Chipkarte benutzt. Um nun seinem Gesprächspartner am Telefon einen Termin zu bestätigen, muss er kurz in seinem persönlichen Terminplaner, der sich ebenfalls auf der Chipkarte befindet, nachsehen. Mit einem zweiten logischen Kanal, parallel zur GSM-Anwendung, sucht das Terminal in einer Datei in der Terminalplan-Anwendung und teilt unserem gestressten Manager mit, ob er zusagen kann oder nicht. Dies wäre eine typische Anwendung von logischen Kanälen. Als weiteres Beispiel wäre auch der sichere Transfer von elektronischen Geldeinheiten zwischen zwei Börsen auf der Chipkarte denkbar.

So nützlich die logischen Kanäle für die Anwendung erscheinen, so problematisch 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 letztendlich 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 noch Secure Messaging und alle Varianten der Authentisierung für jeden logischen Kanal individuell, dann steigt der Speicherbedarf sehr schnell in Bereiche, die nur von den leistungsfähigsten zurzeit existierenden Mikrocontrollern für Chipkarten abgedeckt werden können.

²⁴ Ein Verbesserungsvorschlag für die Überarbeitung der ISO/IEC 7816-3 sieht vor, dass in Zukunft durch zusätzliche Verwendung eines RFU-Bits die Anzahl der möglichen logischen Kanäle auf 8 erhöht wird.

7 Kommandos von Chipkarten

7.1	Kommandos zur Auswahl von Dateien	447
7.2	Schreib- und Lesekommandos	450
7.3	Suchkommandos	457
7.4	Operationen auf Dateien.	459
7.5	Identifizierungskommandos	461
7.6	Authentisierungskommandos	465
7.7	Kommandos für kryptografische Algorithmen	469
7.8	Kommandos zur Verwaltung von Dateien	475
7.9	Kommandos zur Verwaltung von Applets	482
7.10	Kommandos zur Komplettierung des Betriebssystems	482
7.11	Kommandos zum Test der Hardware.	486
7.12	Kommandos für Datenübertragung	489
7.13	Datenbankkommandos – SCQL	491
7.14	Kommandos für elektronische Geldbörsen.	494
7.15	Kommandos für Kredit- und Debitkarten.	497
7.16	Anwendungsspezifische Kommandos	498

Der Kommunikationsablauf eines Terminals mit einer Chipkarte basiert immer auf dem Master-Slave-Prinzip. Dies bedeutet, dass das Terminal ein Kommando als Master zur Chipkarte sendet, die es als Slave daraufhin unverzüglich abarbeitet, eine Antwort erzeugt und sie an das Terminal zurücksendet. Die Karte sendet also nie von sich aus Daten, ohne vorher vom Terminal ein entsprechendes Kommando erhalten zu haben. Selbst der ATR ist keine Ausnahme von dieser Regel, da er die Antwort auf das Reset-Signal ist, das in gewisser Weise auch eine Art von Kommando an die Karte ist.

Die Kommunikation selber läuft immer im Rahmen eines Übertragungsprotokolls ab, also mit $T=0$ oder $T=1$. Diese nicht sehr komplizierten Protokolle erfüllen die speziellen Anforderungen von Chipkarten-Anwendungen und sind auch dahingehend optimiert. Abweichungen von diesen genau festgelegten Protokollen innerhalb des Anwendungsablaufes sind nicht erlaubt. Die Übertragungsprotokolle ermöglichen es, Daten vollständig transparent zur Übertragungsschicht zur Chipkarte hin und wieder zurück zu senden. Diese Daten sind in eine Art Container eingebettet, die Application Protocol Data Unit (APDU) heißen. Die APDUs, die vom Terminal zur Chipkarte gesendet werden, sind die Kommandos an die Karte. Die von ihr daraufhin erzeugten Antworten erhält das Terminal ebenfalls in einer in das Übertragungsprotokoll eingebetteten APDU. Aufbauend auf diesen Mechanismen existiert eine Vielzahl von Kommandos an die Karte, die dann Aktionen in dieser auslösen. Die einfachsten Beispiele sind die Lese- oder Schreibkommandos für Dateien der Chipkarte.

In Chipkarten-Anwendungen benutzt man die Karte entweder als Datenträger, Berechtigungsträger oder beides zugleich. Das führt zu speziell für diese Anforderungen und Übertragungsprotokolle optimierten Befehlssätzen, die auch nur im Chipkartenbereich Verwendung finden. Aufgrund des beschränkten Speicherplatzes der Chipkarten und des Drucks des Marktes, diesen aus Kostengründen nur mäßig zu erweitern, wird der Befehlssatz oft auf eine bestimmte Anwendung zugeschnitten. Alle Kommandos, die innerhalb einer An-

wendung nicht benötigt werden, fallen rigoros der Optimierung bei der Programmierung zum Opfer. Nur wenige Betriebssysteme weisen einen umfangreichen Befehlssatz auf, der nicht für eine bestimmte Anwendung reduziert ist.

Bei den Befehlssätzen für Chipkarten tritt, zusätzlich zu den Optimierungen für bestimmte Anwendungsfälle, noch der bei neuen Technologien typische Diversifizierungseffekt auf. Jede Firma, die im Chipkartenbereich tätig ist, versucht, für ihre Betriebssysteme oder für die zu projektierende Anwendung eigene Kommandos maßzuschneidern. Oft kommt dies aus dem Zwang, dass keine funktionell vergleichbaren Kommandos in den Normen existieren. Oder Firmen versuchen ganz bewusst, sich hinsichtlich Funktionen und Speicherplatzverbrauch mit hochoptimierten Kommandos von Konkurrenzprodukten abzuheben oder diesen damit den Zutritt zu einer bestimmten Anwendung zu verwehren. Allerdings bedeutet eine Entscheidung für Kommandos auf der Grundlage vorhandener Normen immer auch eine Entscheidung für ein offenes, leichter erweiterbares und auch erprobtes System, in dem später vielleicht auch mehrere Anwendungen auf einer Chipkarte zusammengefasst werden können. Es lassen sich jedoch auch genügend Beispiele finden, bei denen der Einsatz von Chipkarten überhaupt erst durch hochoptimierte Spezialkommandos möglich gemacht wurde.

Es gibt zurzeit 13 internationale Normen bzw. mehr oder weniger stabile Entwürfe für diese Normen, in denen typische Kommandos für Chipkarten definiert sind. In diesen Normen sind weit mehr als 100 Kommandos und die dazugehörigen Abläufe aufgeführt. Die definierten Kommandos sind in Codierung und Funktionalität untereinander weitgehend kompatibel.

Ein Großteil der heute in Chipkarten verwendeten Kommandos wird in der allgemeinen und internationalen Industriennorm ISO/IEC 7816-4 festgelegt. Sie widmet sich nicht einem speziellen Bereich, wie etwa der Telekommunikation oder dem Zahlungsverkehr, sondern hat den Anspruch, alle Chipkarten-Anwendungsgebiete abzudecken. Die Kommandos aus ISO/IEC 7816-4 werden durch drei zusätzliche und spezialisiertere Teile der Normenreihe ergänzt. In der ISO/IEC 7816-7 sind Kommandos zur Abfrage und Verwaltung einer an SQL (*structured query language*) angelehnten Datenbankstruktur für Chipkarten festgelegt. Kommandos zur Ausführung und Parametrisierung von kryptografischen Funktionen enthält die Norm ISO/IEC 7816-8. Teil 9 der ISO/IEC 7816 Normenreihe ergänzt den Befehlssatz um Dateiverwaltungskommandos.

In dem Bereich Zahlungsverkehr gibt es keine internationale Norm von Bedeutung, sondern einen Industriestandard. Dies ist der EMV-Standard, der sich aus den drei Anfangsbuchstaben Europay, Mastercard und Visa der drei Initiatoren dieses Standards ableitet. Diese Norm ist aufgrund der Marktmacht der beteiligten Firmen für alle Chipkarten-Betriebssysteme zur Referenz aufgestiegen und hat eine ähnliche Bedeutung wie die ISO/IEC 7816 Normenreihe.

Die für den Einsatz im Bereich der Telekommunikation entwickelte Norm GSM 11.11 stellt die normative Grundlage für die SIM dar und die Normen TS 31.102, 31.111 und TS 102.222 legen die grundlegenden Kommandos für die USIM fest. Alleine aufgrund der enormen Verbreitung von Telekommunikations-Chipkarten definieren diese Normen einen weltweiten *de facto*-Standard für Kommandos von Chipkarten-Betriebssystemen.

Spezialkommandos, die nur in einem begrenzten Umfeld Einsatz finden, fallen prinzipiell nicht unter diese Normen und müssen deshalb individuell festgelegt werden. Ein Beispiel dafür ist der festgelegte Befehlssatz für eine branchenübergreifende elektronische Geldbörse, der in der CEN-Norm EN 1546 definiert ist. Dies ist eine europäische Norm, in der alle für eine elektronische Geldbörse notwendigen Kommandos und auch die Abläufe

dazu enthalten sind. Diese auf eine einzige Anwendung beschränkten Normen entstehen nur für Gebiete, die stark im Interesse von Behörden oder bestimmten Industriezweigen stehen, da der Aufwand dafür sonst viel zu groß wäre.

Die einzelnen Kommandos der obig aufgeführten Normen lassen sich anhand ihrer Funktionalität in Klassen ordnen. Es sollte jedoch bedacht werden, dass in real existierenden Chipkarten-Betriebssystemen immer nur eine Untermenge aller in diesen Klassen gesammelten Kommandos implementiert ist. Je nach Betriebssystem-Hersteller können mehr oder weniger große Abweichungen in der in den folgenden Absätzen beschriebenen Funktionalität und auch in der Codierung auftreten. Die aufgeführten Grundfunktionen sind aber im Prinzip in allen Betriebssystemen vorhanden. Aus Speicherplatz- und Kostengründen kann allerdings auch die Funktionalität stark eingeschränkt sein. Bei einer zu projektierenden Anwendung muss auf jeden Fall die genaue Spezifikation jedes einzelnen Kommandos in Codierung und Funktion vom jeweiligen Betriebssystemhersteller erfragt werden.

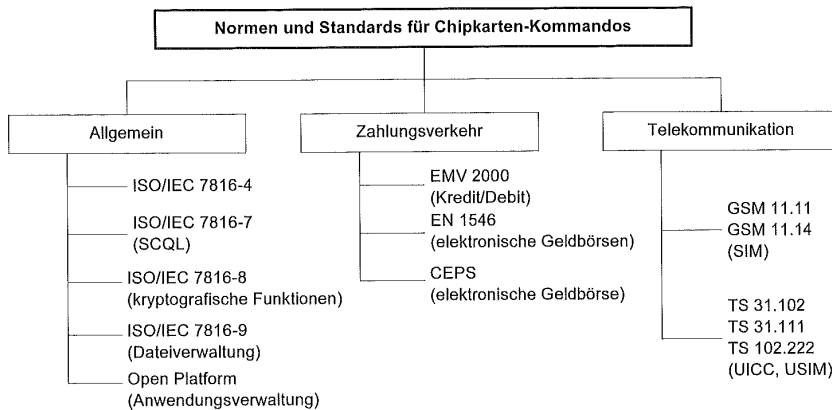


Bild 7.1 Die wichtigsten Normen und Industriestandards für Chipkarten-Kommandos.

In den nachfolgenden Abschnitten sind die wichtigsten und am weitesten verbreiteten Kommandos für Chipkarten aufgeführt. Die Grundlage für die Auswahl waren die Normen ISO/IEC 7816-4, -7, -8, -9, EMV 2000, GSM 11.11, GSM 11.14, TS 31.102, TS 31.111, EN 726-3 und EN 1546-3. Umfangreiche Tabellen mit der Codierung der wichtigsten Chipkarten-Kommandos finden sich unter 16.11.7 Codierung von Chipkarten-Kommandos.

Es gibt allerdings weltweit keine einzige Chipkarte zu kaufen, die alle nachfolgenden Kommandos enthält. Der Speicherbedarf für die vollständige Implementierung dürfte, vorsichtig geschätzt, zwischen fünf- und zehnmal höher liegen, als die größten zurzeit erhältlichen Chipkarten-Mikrocontroller als Gesamtspeicher anbieten. Es ist aber auch gar nicht notwendig, dass eine Chipkarte alle diese Kommandos ausführen kann. Je nach projektiertem Einsatzgebiet und Betriebssystem können mal jene oder mal andere Kommandoklassen stärker unterstützt sein.

Bei einer Multiapplikations-Chipkarte wird man sicherlich stark darauf achten, dass nach der Personalisierung noch nachträglich Anwendungen in der Karte installiert werden können. Eine Chipkarte, deren Einsatzgebiet im Verschlüsselungsbereich liegt, besitzt,

wenn genügend Speicher vorhanden ist, das volle Spektrum der kryptografischen Kommandos mit den verschiedenen Algorithmen. Jedes Einsatzgebiet bedingt eine unterschiedliche Auswahl der zu verwendenden Kommandos aus den verschiedenen Klassen.

Um den Überblick zu wahren, ist im Folgenden bei jeder Kommandobeschreibung angegeben, welcher Norm das Kommando zuzuordnen ist. Bei nicht genannter Quelle ist es ein Kommando, das bei Chipkartenfirmen in verschiedener Form intern Verwendung findet, jedoch keiner der vorgenannten Normen zuzuordnen ist. Manche dieser Kommandos sind aber sehr nützlich und werden wohl in Zukunft Eingang in die Normung finden. Deshalb sind sie hier aufgeführt und auch in ihrer grundlegenden Funktionalität beschrieben. Auf eine Beschreibung der Codierung von typischen Chipkarten-Kommandos haben wir aus Gründen der Leserlichkeit in diesem Kapitel verzichtet. Diese ist in den Anhang unter „15.8.7 Codierungen von Chipkarten-Kommandos“ ausgelagert.

Es existieren einige Kommandos, die von beinahe allen Chipkarten-Betriebssystemen unterstützt werden und bei denen die Zahl der möglichen Optionen gering ist. Die APDU von Kommando und Antwort ist an der entsprechenden Stelle vollständig decodiert. Den internen Programmablauf von sieben typischen Kommandos kann man in Kapitel 5 anhand des Pseudocodes des einfachen Chipkarten-Betriebssystems „Small-OS“ im Detail nachvollziehen.

Ausgehend von einer Anwendung lassen sich Chipkarten-Kommandos in operative, d. h. für den laufenden Betrieb notwendige, und administrative, d. h. für die Verwaltung

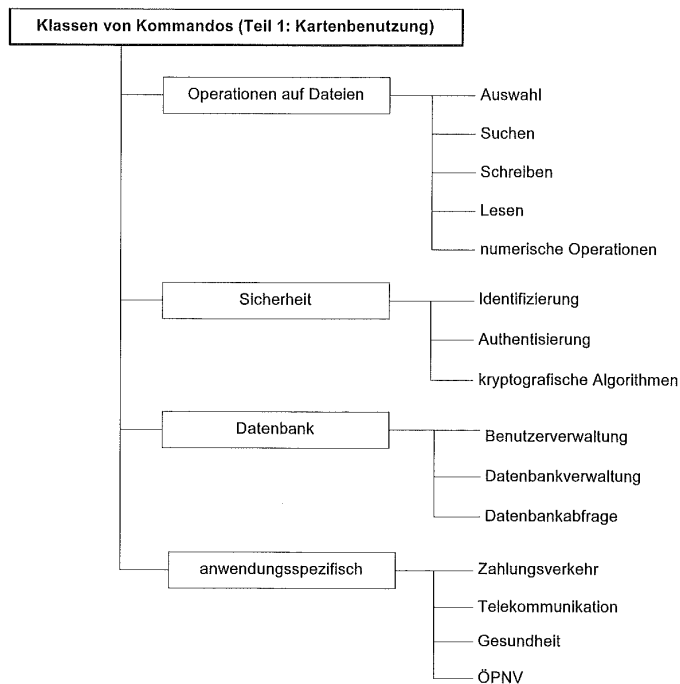


Bild 7.2 Klassifizierungsbaum aller Kommandos von Chipkarten, die vor allem während der Kartenbenutzung, d. h. in der operativen Phase, eingesetzt werden.

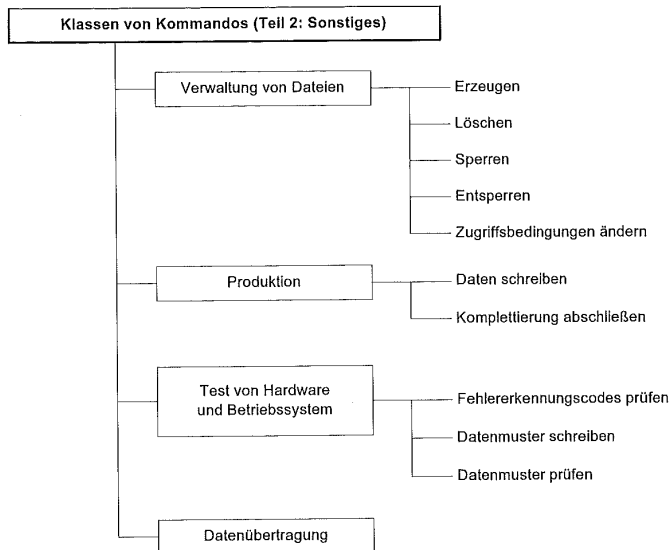


Bild 7.3 Klassifizierungsbaum aller Kommandos von Chipkarten, die hauptsächlich vor und nach der Kartenbenutzung, d. h. in der administrativen Phase, eingesetzt werden.

der Chipkarte und Anwendung notwendige, Kommandos einteilen. Die Kommandos der operativen Phase sind i. d. R. aus Gründen der Interoperabilität durch Normen in allen Details festgelegt. Die Kommandos der Administrativen Phase sind oftmals Betriebssystemspezifisch und vielfach nicht durch Normen festgelegt.

Als Antwort der Chipkarte ist bei der jeweiligen Kommandobeschreibung immer der Gutfall angenommen. Im Schlechtfall, d. h. falls eine Operation verboten ist oder Fehler in der Chipkarte auftraten, erhält das Terminal nur einen 2 Byte langen Returncode zurück. Manche der beschriebenen Kommandos besitzen noch Parameter zur Auswahl von weiteren Funktionen. Diese Varianten existieren oft lediglich in den Normen, jedoch nicht in realen Betriebssystemen, da sie entweder zu aufwendig oder für die Praxis ohne Bedeutung sind. Hier sind deshalb nicht alle Varianten der in den drei Normen beschriebenen Kommandos aufgezählt und erklärt, da ein deutlicher Bezug auf eine für die Praxis relevante Funktionalität hergestellt werden soll. Meistens fand in der Kommandobeschreibung die Norm Verwendung, bei der die funktionellen Möglichkeiten eines Kommandos am größten sind.

7.1 Kommandos zur Auswahl von Dateien

Die Dateiverwaltungen aller neuen Chipkarten-Betriebssysteme sind ausnahmslos objektorientiert aufgebaut. Dies bedeutet unter anderem, dass vor einer Aktion an einem Objekt (welches einer Datei entspricht) dieses zuerst ausgewählt werden muss. Erst dann ist dem Betriebssystem bekannt, welche Datei gemeint ist, und alle nachfolgenden dateispezifischen Kommandos betreffen nur diese eine Datei. Es müssen selbstverständlich innerhalb des Betriebssystems dann noch die Zugriffsbedingungen zu der Datei überprüft werden, d. h. ob die betreffenden Kommandos überhaupt erlaubt bzw. möglich sind.

Nach dem Reset einer Chipkarte ist das MF immer implizit selektiert und muss also nicht noch extra ausgewählt werden. Die anschließende Auswahl von Dateien wird mit dem Kommando SELECT FILE durchgeführt. Die Adressierung findet entweder mit dem 2 Byte langen FID (*file identifier*) oder zusätzlich im Falle von DFs mit dem 1 bis 16 Byte langen DF Name statt. Dieser kann einen 5 bis 16 Byte langen und international eindeutigen AID (*application identifier*) enthalten. Es ist möglich, den AID auch nur teilweise zu übergeben. Dazu werden die niederwertigsten Bytes (d. h. die rechts stehenden) im AID weggelassen. Durch einen zusätzlichen Parameter selektiert die Karte das erste, letzte, nächste oder das vorhergehende (*first, last, next, previous*) DF, auf das der gekürzte AID zutrifft.

Im Rahmen der älteren Kommandodefinition bei GSM 11.11 ist die Auswahl einer Datei nur mit dem 2 Byte langen FID möglich. Der ISO Befehlssatz hingegen unterstützt zusätzlich als Erweiterung auch noch eine Dateiauswahl durch die Pfadangabe zur betreffenden Datei. Die Pfadangabe kann entweder relativ sein, dann erfolgt sie vom aktuell selektierten DF aus oder kann auch durch eine absolute Angabe des Pfades zur gewünschten Datei vom MF aus erfolgen.

Nur eine erfolgreiche Auswahl einer neuen Datei bewirkt die Deselektion der ursprünglich ausgewählten Datei. Konnte die Selektion nicht durchgeführt werden, weil z. B. die Datei nicht existiert, dann bleibt die vorhergehende Auswahl bestehen. Damit ist im Fehlerfall sichergestellt, dass immer eine Datei selektiert ist.

Nach einer erfolgreichen Selektion können bei Bedarf vom Terminal Informationen über die neue aktuelle Datei angefordert werden. Dies und die Anzahl der gewünschten Informationen über die Datei erhält die Chipkarte im Rahmen des SELECT FILE Kommandos. Definiert ist der exakte Inhalt dieser Informationsdaten in den jeweiligen Normen. Die von der Karte dabei gesendeten Informationen können unter anderem Struktur, Größe und noch vorhandenen Speicher der neu ausgewählten Datei umfassen. Auch kann der Umfang dieser Informationen noch zusätzlich vom Dateityp abhängig sein.

Die folgenden Möglichkeiten der expliziten Selektion von Dateien sind mit dem SELECT FILE-Kommando nach ISO/IEC 7816-4 erlaubt.

Tabelle 7.1 Die Funktionalität von SELECT FILE nach ISO/IEC 7816-4

SELECT FILE	
Kommando	- FID (falls EF, DF, MF) oder DF Name (falls DF) oder Pfad zur Datei vom aktuell ausgewählten DF oder Pfad zur Datei vom MF aus oder Schalter: Selektiere übergeordnetes DF oder erstes, letztes, nächstes, vorheriges DF (falls eine Teil-AID übergeben wurde) - Schalter: Informationen über ausgewählte Datei zurücksenden
Antwort	- Informationen über die ausgewählte Datei (falls via Schalter gewählt) - Returncode

Das typische Beispiel der Abfolge bei der Auswahl einer Datei ist in dem folgenden Ablauf dargestellt:

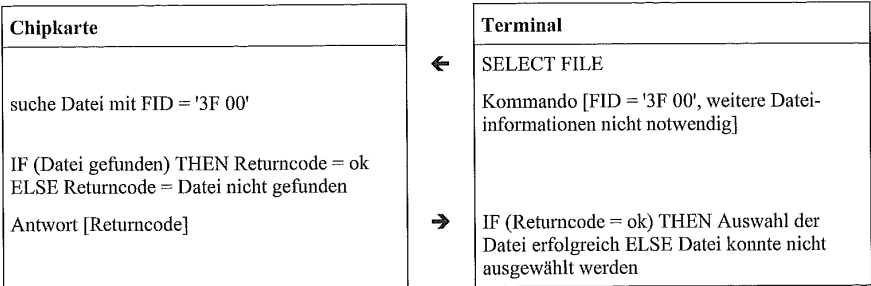


Bild 7.4 Beispiel für den Kommandoablauf bei einem SELECT FILE Kommando.

Zusätzlich zu dieser expliziten Auswahl einer Datei durch Angabe von FID, DF Name oder des Pfades im Kommando SELECT FILE existiert die Möglichkeit einer impliziten Auswahl. Dies ist aber nur mit den Standard-Lese- und Schreib-Kommandos möglich. Unter Angabe einer 5 Bit langen Short-FID in Form eines zusätzlichen Parameters kann vor der eigentlichen Kommandoausführung noch eine Datei gewählt werden. Die Datei muss aber vom Typ EF sein und sich innerhalb des aktuellen DF befinden. Der Vorteil dieses Verfahrens liegt in einem vereinfachten Kommandoablauf und einer Erhöhung der Ablaufgeschwindigkeit, da kein explizites SELECT FILE-Kommando mehr zur Karte übertragen werden muss.¹

In der GSM 11.11 Norm ist ein Kommando definiert, das die gleichen Daten an das Terminal zurückgibt wie eine erfolgreiche Auswahl einer Datei mit SELECT FILE. Dies sind Informationen über Typ und Struktur der aktuell selektierten Datei, Dateigröße, FID, Zugriffsbedingungen und den Zustand, ob die Datei gesperrt ist. Dieses Kommando wird selten benutzt und hat vor allem den Zweck, dass ein Terminal während einer Sitzung anfragen kann, welche Datei einschließlich diverser Rahmenbedingungen zurzeit aktuell selektiert ist.

Tabelle 7.2 Die Funktionalität von STATUS nach GSM 11.11

STATUS	
Kommando	• ---
Antwort	• Informationen über die aktuell ausgewählte Datei
	• Returncode

In Ergänzung zum SELECT FILE- und STATUS-Kommando gibt es in der EN 726-3 ein Kommando zum Schließen von Anwendungen. Das Kommando CLOSE APPLICATION übergibt die FID der zu schließenden Anwendung, und daraufhin wird in der Chipkarte der erreichte Sicherheitszustand gelöscht. Nützlich ist dieses Kommando vor allem

¹ Siehe auch Abschnitt 5.6.2 „Dateinamen“.

dann, wenn ein Terminal sicherstellen will, dass ein erreichter Zustand in der Chipkarte wieder zurückgesetzt wird. Unterstützt das Betriebssystem einer Chipkarte dieses oder ein ähnliches Kommando nicht, so ist zum Rücksetzen des Zustands ein Reset der Karte notwendig. In der Definition von ISO/IEC 7816-4 reicht die Selektion des MF aus, um die Sicherheitszustände des vorher selektierten DF auf ihren Initialzustand zurückzusetzen.

Tabelle 7.3 Die Funktionalität von CLOSE APPLICATION nach EN 726-3

CLOSE APPLICATION	
Kommando	• FID (des aktuellen DFs)
Antwort	• Returncode

7.2 Schreib- und Lesekommandos

Die Klasse der Schreib- und Lesekommandos unterstützt vor allem die Verwendung einer Chipkarte als sicheren Datenspeicher. Mit diesen Kommandos lassen sich Informationen indie betreffenden EFs schreiben und später auch wieder auslesen. Sofern die EFs bestimmte Zugriffsbedingungen haben, ist das Lesen dieser Dateien nur noch für autorisierte Benutzer möglich. Die Information ist also sicher gegen unbefugten Zugriff in der Chipkarte gespeichert.

Da für EFs unterschiedliche Datenstrukturen existieren, gibt es auch unterschiedliche Schreib- und Lesekommandos für diese Dateien. Dies entspricht leider nicht ganz einem objektorientierten Aufbau einer Dateiverwaltung. Bei einer reinen Objektorientierung müsste das Betriebssystem so gestaltet sein, dass ein Objekt selber seine Zugriffsmechanismen bestimmen kann. Dies ist aber bei Dateiverwaltungen für Chipkarten nicht der Fall. Zurückzuführen ist diese Abweichung auf historisch gewachsene Kommandos, die in aktuelle Normen übernommen wurden. Die Vorgänger der Chipkarten, die Speicherkarten, haben nur einen einzigen Dateibereich, der mit Offset- und Längenangabe schreib- und lesbar ist. Nach außen hin kann man diesen Speicher als eine einzige Datei mit transparenter Struktur ansehen. Die ersten Chipkarten waren im Prinzip genauso aufgebaut, und eben aus dieser Zeit stammen die Definitionen der Kommandos für Schreib- und Lesezugriffe auf transparente Dateien. Für die später definierten Dateien mit Recordstruktur verwendete man andere, speziell angepasste Kommandos. So entstanden die beiden unterschiedlichen Zugriffstypen von Kommandos.

So muss man diese Klasse in Kommandos einteilen, die auf EFs mit transparenter Struktur zugreifen und in Zugriffskommandos für die übrigen Dateistrukturen cyclic, linear fixed und linear variable. Teilweise ist aber in verschiedenen Normen (z. B. elektronische Geldbörse nach EN 1546) noch explizit angegeben, dass es erlaubt ist, mit einem Lesekommando für Dateien mit transparenter Struktur auch Dateien mit anderer Struktur zu lesen. Allerdings kann man dabei zusätzliche Informationen über die interne Struktur der Datei erhalten.

EFs mit der logischen Struktur transparent sind amorph aufgebaut, besitzen also keine innere Struktur. Sie stellen einen linear adressierbaren Speicher dar, auf den byteweise zugegriffen werden kann. Dabei verwendet man das Kommando READ BINARY zum Lesen und die beiden Kommandos WRITE BINARY und UPDATE BINARY zum Schreiben.

Tabelle 7.4 Die Funktionalität von READ BINARY nach ISO/IEC 7816-4

READ BINARY	
Kommando	- Anzahl der zu lesenden Bytes - Offset zum ersten Byte, das gelesen werden soll <i>ptional</i>: Short-FID für implizite Selektion
Antwort	- gelesene Daten aus der Datei - Returncode

Tabelle 7.5 Die Funktionalität von WRITE BINARY nach ISO/IEC 7816-4

WRITE BINARY	
Kommando	- Anzahl der zu schreibenden Bytes - zu schreibende Bytes - Offset zum ersten Byte, das geschrieben werden soll <i>optional</i>: Short-FID für implizite Selektion
Antwort	Returncode

Der grundsätzliche Unterschied zwischen den Schreibkommandos WRITE und UPDATE ist in Hinblick auf den sicheren Zustand des EEPROM der Chipkarte zu sehen. Der sichere EEPROM-Zustand ist der logische Wert der Bits im EEPROM, bei dem die Speicherzellen den energieärmsten Zustand eingenommen haben, d. h., wenn die EEPROM-Zellen, die kleine Kondensatoren sind, keine Ladung mehr haben. Üblicherweise ist dies der logische Zustand 0. Um nun ein Bit vom Zustand 0 wieder in den Zustand 1 zu setzen, muss es gelöscht werden. Dadurch bringt man wieder Ladung auf den Kondensator auf.

Mit WRITE kann man nun die Bits nur vom unsicheren Zustand, im Beispiel der Zustand 1, in den sicheren Zustand, d. h. in diesem Fall auf 0 setzen. WRITE ist also in diesem Beispiel eine logische UND-Verknüpfung der übergebenen Daten mit dem Dateiinhalt. Falls der sichere Zustand des Chips durch den Wert 1 repräsentiert wird, muss WRITE eine logische ODER-Verknüpfung der Daten im Kommando zu den Daten in der Datei bewirken. Die logische Verknüpfung zwischen den Daten im Kommando und den Daten in der Datei ist dabei immer so, dass bei WRITE der sichere Zustand des EEPROMs erreicht wird. Zusätzlich kann WRITE noch abhängig von der Datei einen WORM-Schreibzugriff (*write once, read multiple*) unterstützen. WRITE Kommandos stammen aus einer Zeit, in der atomare Zugriffe bei Chipkarten noch unbekannt waren. Sie werden heute nur noch sehr selten verwendet.

Das Kommando UPDATE ist hingegen ein echtes Schreiben auf den Dateiinhalt. Die Daten, die vorher darin enthalten waren, haben keinerlei Einfluss auf den Inhalt der Datei nach dem UPDATE-Kommando. UPDATE kann deshalb auch als vorheriges Löschen mit ERASE und anschließendes Schreiben mit WRITE betrachtet werden.

Tabelle 7.6 Die Funktionalität von UPDATE BINARY nach ISO/IEC 7816-4

UPDATE BINARY	
Kommando	- Anzahl der zu überschreibenden Bytes - zu überschreibende Bytes - Offset zum ersten Byte, das überschrieben werden soll <i>optional</i>: Short-FID für implizite Selektion
Antwort	Returncode

Mit diesen beiden Kommandos kann man physikalisch sichere Zähler in Chipkarten aufbauen. Das Prinzip dabei ist ein Bitfeld, bei dem gesetzte Bits jeweils eine Geldeinheit verkörpern. Bei der Zahlung erniedrigt man mit WRITE den Zähler Bit für Bit durch die ODER-Verknüpfung. Nach einer Authentisierung kann man den Zähler mit UPDATE wieder erhöhen. Der große Vorteil dieses Aufbaus ist, dass zum Beispiel durch thermische Beeinflussung des EEPROMs dieses nicht auf einen höheren Zählerstand gebracht werden kann, da der sichere EEPROM-Zustand des Bitfeldes den Wert 0 symbolisiert.

Wie der Name schon aussagt, stellt READ BINARY ein Lesekommando dar, und WRITE/UPDATE BINARY wird zum Schreiben von Daten verwendet. Der Zugriff auf eine Datei wird dabei durch eine Längenangabe und einen Offset bis zum ersten anzusprechenden Byte realisiert. Manche Betriebssysteme ermöglichen zusätzlich vor dem eigentlichen Dateizugriff die implizite Auswahl einer Datei durch Übergabe eines Short-FID. Dies ist jedoch noch nicht in allen Normen und Betriebssystemen vorgesehen.

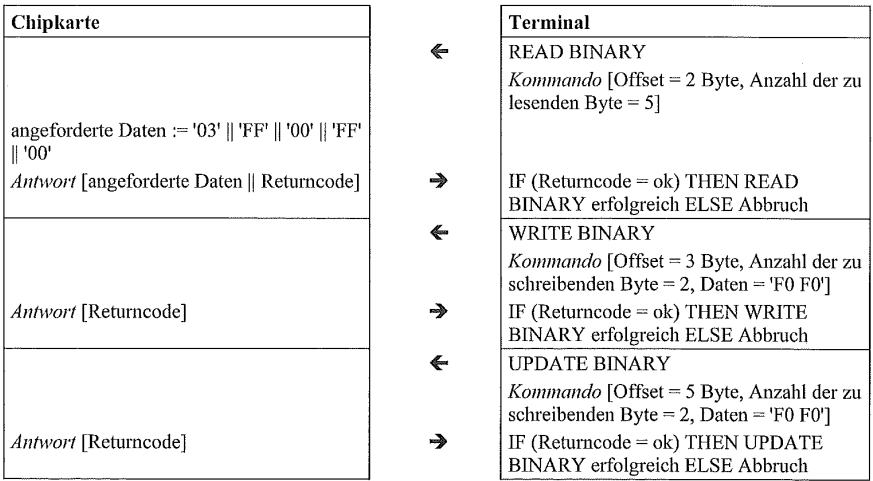


Bild 7.5 Zugriff auf eine Datei mit transparenter Struktur.

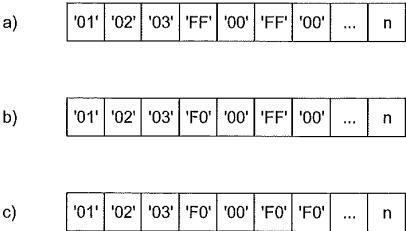


Bild 7.6 Beispiel für Schreibzugriffe auf ein EF mit transparenter Struktur. Die dazu verwendeten Kommandosequenzen sind in Bild 7.5 angegeben.

- a) Dateiinhalt bei READ BINARY
- b) Dateiinhalt nach WRITE BINARY
- c) Dateiinhalt nach UPDATE BINARY

Bild 7.5 zeigt einen Ablauf mit den Kommandos READ BINARY, eines darauf folgenden WRITE BINARY und abschließenden UPDATE BINARY. Die Auswirkungen auf den Dateinhalt der selektierten Datei sind in Bild 7.6 zu sehen. Die Kommandos und Abläufe setzen selbstverständlich eine erfolgreiche Selektion und Erfüllung aller Zugriffsbedingungen auf die beschriebene Datei voraus.

Eine Ausnahme bei den an transparenten EFs orientierten Kommandos stellt ERASE BINARY dar. Mit diesem Kommando können keine Daten in eine Datei geschrieben werden. Er dient lediglich dazu, Dateiinhalte ab einem zu übergebenden Offset zu löschen. Falls kein zweiter Offset bis zum Ende des zu löschenden Bereichs mit übergeben wird, dann löscht das Kommando bis zum Ende der ausgewählten Datei. Löschen heißt in diesem Zusammenhang, dass der im Kommando angegebene Datenbereich auf den logisch gelöschten Zustand gesetzt wird. Dieser muss individuell für jedes Betriebssystem definiert werden, da er unterschiedlich zum physikalisch gelöschten Zustand des Speichers sein kann.

Aufgrund der gänzlich anderen Struktur von linear fixed, linear variable und cyclic EFs existieren parallel zu den obigen Kommandos noch besondere Kommandos für Zugriffe auf diese Datenstrukturen. Alle diese Dateien sind Record-orientiert aufgebaut. Die kleinste ansprechbare Einheit im Datenfeld beim Schreibzugriff ist ein einzelner Record. Beim lesenden Zugriff kann entweder der ganze Record oder ein Teil von ihm ab dem ersten Byte gelesen werden.

Durch diese Dateistrukturen, die den linearen 1-dimensionalen Speicher in einen 2-dimensional ansprechbaren Speicher transformieren, ergeben sich im Gegensatz zu den transparenten Strukturen auch wesentlich komplexere Zugriffsarten. Prinzipiell können aber alle Datenstrukturen durch Dateien der Struktur transparent nachgebildet werden. Dies kann jedoch im Einzelfall erheblich aufwendiger sein als bei der Verwendung von höheren Datenstrukturen.

Nach der Auswahl eines EF mit Record-orientierter Struktur erzeugt das Chipkarten-Betriebssystem einen Recordzeiger, dessen Wert anfangs undefiniert ist und durch READ/WRITE/UPDATE RECORD oder SEEK gesetzt werden kann. Der Zeiger für die aktuelle Datei bleibt aber erhalten, solange diese selektiert ist. Nach einer erfolgreichen expliziten oder impliziten Selektion einer anderen Datei erhält der Recordzeiger wieder einen undefinierten Wert.

Tabelle 7.7 Die Funktionalität von ERASE BINARY nach ISO/IEC 7816-4

ERASE BINARY	
Kommando	• Offset zum ersten Byte, das gelöscht werden soll <i>optional</i>: • Offset zum letzten Byte, das gelöscht werden soll • <i>optional</i>: Short-FID für implizite Selektion
Antwort	• Returncode

Alle Kommandos für Record-orientierte Dateien können über ein Parameterbyte die Art der Zugriffe auf den Dateinhalt bestimmen. Die Grundlage bildet dabei der direkte Zugriff unter Angabe der absoluten Nummer des gewünschten Records. Dieser Zugriff verändert

nicht den Recordzeiger. Man teilt also der Chipkarte die Nummer des gewünschten Records mit und erhält als Antwort den Inhalt des betreffenden Records.

Durch Angabe des Parameters „first“ setzt das Betriebssystem den Recordzeiger auf den ersten Record der Datei, und dieser wird dann je nach Kommando gelesen oder geschrieben. Analog dazu existiert für den Zugriff auf den letzten Record der Parameter „last“. Mit den zusätzlichen Parameterangaben „next“ und „previous“ lassen sich der nächste bzw. der vorherige Record auswählen und lesen bzw. schreiben. Als Letzter der möglichen Parameter kann „current“ benützt werden, um den Record anzusprechen, auf den der Recordzeiger aktuell ausgerichtet ist.

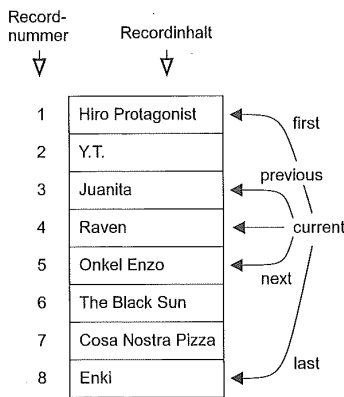


Bild 7.7 Zugriff auf eine Datei mit Record-orientierter Struktur.

Diese sehr große Variabilität des Zugriffs auf Record-orientierte Datenstrukturen hat ihre Ursache in dem typischen Aufbau von Telefonnummernverzeichnissen. Man stelle sich einen Record vor, dessen Anfangsteil Vor- und Nachname enthält und anschließend im gleichen Record die dazugehörige Telefonnummer.

Name	Telefonnummer
Joshua Calvert	47 84 46
Quinn Dexter	089 11 00 11
Louise Kavanagh	089 47 51 22
Alkad Mzu	089 178 098
Ione Saldana	08933 178 234
Kiera Salter	089 23 76 87 33
Ralph Hiltch	089 78 123 78 1
Fletcher Christian	089 12 111 222
Kelly Tirrel	089 92 178 234
Gerald Skibbow	089 129 167 189

Bild 7.8 Beispiel für eine Telefonnummernliste in einer Datei der Struktur „linear fixed“.

Mit einem READ RECORD und den obig erläuterten Parametern kann man je nach Bedarf innerhalb dieser auf ein EF abgebildeten Telefonliste vor- und zurückblättern oder auf den ersten oder letzten Eintrag springen. Im Übrigen ist es auch möglich, den Recordzeiger durch das Suchkommando SEEK, das im folgenden Abschnitt vorgestellt ist, zu verändern.

Nach ISO/IEC 7816-4 besteht auch die Möglichkeit, alle Records von der Ersten bis zu einer zu übergebenden Recordnummer auszulesen. Analog dazu lassen sich auch alle Records ab einem bestimmten bis zum Letzten mit READ RECORD in einem einzigen Kommando-Antwort-Paar aus der Datei lesen. So praktisch diese Kommandos sind, kann mit ihnen bei größeren Dateien jedoch sehr schnell die Grenze des I/O-Puffers der Chipkarte erreicht werden.

Tabelle 7.8 Die Funktionalität von READ RECORD nach ISO/IEC 7816-4

READ RECORD	
Kommando	• Nummer des zu lesenden Records <i>oder</i> Modus (erster, letzter, nächster, vorheriger Record) <i>oder</i> Lies alle Records ab n bis zum letzten Record <i>oder</i> Lies alle Records vom ersten Record bis n • <i>optional</i>: Short-FID für implizite Selektion
Antwort	• gelesene Daten aus der Datei • Returncode

Tabelle 7.9 Die Funktionalität von WRITE RECORD nach ISO/IEC 7816-4

WRITE RECORD	
Kommando	• zu schreibender Record Nummer des zu schreibenden Records <i>oder</i> Modus (erster, letzter, nächster, vorheriger Record) • <i>optional</i>: Short-FID für implizite Selektion
Antwort	• Returncode

Tabelle 7.10 Die Funktionalität von UPDATE RECORD nach ISO/IEC 7816-4

UPDATE RECORD	
Kommando	• zu überschreibender Record • Nummer des zu überschreibenden Records <i>oder</i> Modus (erster, letzter, nächster, vorheriger Record) • <i>optional</i>: Short-FID für implizite Selektion
Antwort	• Returncode

Die folgenden Kommandoabläufe zeigen beispielhaft einige Lese- und Schreiboperationen auf die Datei von Bild 7.8.

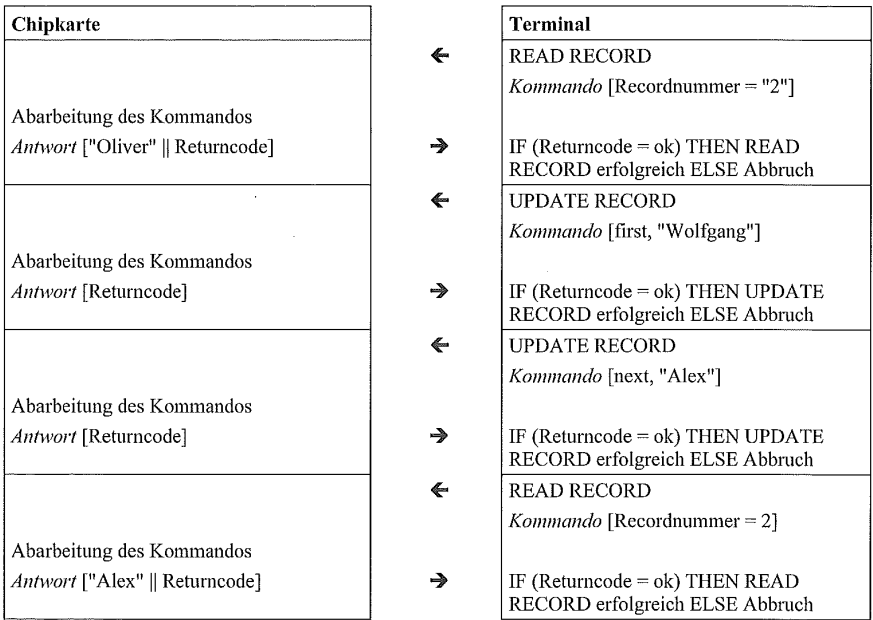


Bild 7.9 Beispiel für Lese- und Schreiboperationen auf Record-orientierte Dateistrukturen.

Das Kommando APPEND RECORD könnte aufgrund seiner Funktion auch genauso im Unterkapitel „Verwaltung von Dateien“ stehen. Mit ihm können an vorhandene Record-orientierte Dateien zusätzliche Records angehängt werden. Gleichzeitig werden im Kommando die Daten für den gesamten neuen Record übergeben. Voraussetzung für das Vorhandensein dieses Kommandos in seiner vollen Funktionalität ist eine für Chipkarten relativ aufwendige Speicherverwaltung. Diese hat dafür zu sorgen, dass eine Verbindung des neuen Records zu den bereits vorhandenen Records hergestellt wird. Dann ist es möglich, im Rahmen des zur Verfügung stehenden Speichers beliebig viele neue Records in einer Datei zu ergänzen. Oftmals behilft man sich aber damit, dass man nur eine bestimmte Anzahl von neuen Records zulässt. Bei der Erzeugung von Record-orientierten Dateien wird dabei je nach Bedarf schon Speicherplatz für die noch nicht erzeugten Records reserviert. Dieser wird dann mit APPEND RECORD aufgefüllt. Ist dieser Platz verbraucht, sind weitere APPEND RECORD-Kommandos nicht mehr möglich.

Benutzt man APPEND RECORD in Verbindung mit einer Datei, die eine „linear fixed“- oder „linear variable“- Struktur ausweist, wird der neue Record immer am Ende der Datei angelegt. Ist die Struktur der Datei aber „cyclic“, hat der neue Record immer die Nummer 1, was im Übrigen auch dem aktuell geschriebenen Record bei Dateien dieser Struktur entspricht.

APPEND RECORD kann für verschiedene Zwecke eingesetzt werden. Denkbar ist die schon mehrfach erwähnte Telefonnummernliste oder eine Protokolldatei, bei der die zu protokollierenden Daten unmittelbar bei der Erzeugung des neuen Records in die Karte geschrieben werden.

Tabelle 7.11 Die Funktionalität von APPEND RECORD nach ISO/IEC 7816-4

APPEND RECORD	
Kommando	• zu schreibender Record • <i>optional</i>: Short-FID für implizite Selektion
Antwort	• Returncode

Als Ergänzung der auf Dateien basierenden Lese- und Schreibkommandos gibt es zwei Kommandos, die für den direkten Zugriff auf Datenobjekte gedacht sind. Je nach ausgewähltem DF können damit vorbei an den dateiorientierten Zugriffsmechanismen bestimmte Daten sowohl in Dateien oder auch in interne Betriebssystemstrukturen geschrieben oder aus diesen Bereichen gelesen werden. Mit PUT DATA können Datenobjekte geschrieben werden. Zum Lesen dieser Daten dient das Kommando GET DATA. Bei beiden Kommandos muss die genaue Struktur der TLV-codierten Datenobjekte übergeben werden, d. h., es muss bekannt sein, ob eine anwendungsspezifische oder eine genormte Codierung der Datenobjekte verwendet wird. Diese Kenntnis ist intern für das Betriebssystem wichtig, damit die Daten entsprechend ihrer Verpackung durch die Objekte erkannt werden. Die entsprechend festgelegten Zugriffsbedingungen müssen sowohl bei PUT DATA als auch bei GET DATA vorab erfüllt sein.

Tabelle 7.12 Die Funktionalität von GET DATA nach ISO/IEC 7816-4

GET DATA	
Kommando	• Anzahl der zu lesenden Datenobjekte • Kennzeichen der zu lesenden Datenobjekte
Antwort	• gelesene Datenobjekte • Returncode

Tabelle 7.13 Die Funktionalität von PUT DATA nach ISO/IEC 7816-4

PUT DATA	
Kommando	• Struktur der zu schreibenden Datenobjekte • zu schreibende Datenobjekte
Antwort	• Returncode

7.3 Suchkommandos

Record-orientierte Strukturen eignen sich gut dafür, zusammengehörige Informationen gleichen Aufbaus in einer Datei abzulegen. Das typische Beispiel dafür ist ein Telefonverzeichnis mit Namen und Telefonnummer. Um nicht bei der Suche nach einem bestimmten Namen das gesamte Verzeichnis Record für Record lesen zu müssen, kann dafür ein Suchkommando benutzt werden.

Mit dem Kommando SEEK kann man mit einem Offset in einer Record-orientierten Datenstruktur nach einer zu übergebenden Zeichenkette zu suchen. Die Länge des Suchstrings ist dabei variabel. Obligatorisch ist im Kommando dem Betriebssystem noch die Suchrichtung mitzuteilen. Das bedeutet, ob nun vorwärts (zu höheren Recordnummern hin) oder rückwärts (zu niedrigeren Recordnummern hin) gesucht werden soll. Auch ist die Startposition des Suchvorgangs anzugeben. Dabei existieren die Möglichkeiten erster Record, letzter Record und aktueller Record. Konnte die Zeichenkette gefunden werden, setzt das Betriebssystem den Recordzeiger auf diesen Eintrag und teilt die erfolgreiche Suche dem Terminal mit.



Bild 7.10 Suchen innerhalb einer Datei mit Record-orientierter Struktur

Tabelle 7.14 Die Funktionalität von SEEK nach GSM 11.11

SEEK	
Kommando	• Länge des Suchstrings • Suchstring • Offset • Modus (vom Anfang an vorwärts, vom Ende aus rückwärts, ab der nächsten Position vorwärts, ab der vorherigen Position rückwärts) • <i>Schalter</i>: Recordnummer des gefundenen Records zurücksenden
Antwort	• Recordnummer (falls via Schalter gewählt) • Returncode

Der folgende Kommandoablauf zeigt einige Möglichkeiten der Anwendung eines SEEK-Kommandos. Die Grundlage ist dabei die im Bild 7.8 dargestellte Datei mit „linear fixed“-Struktur.

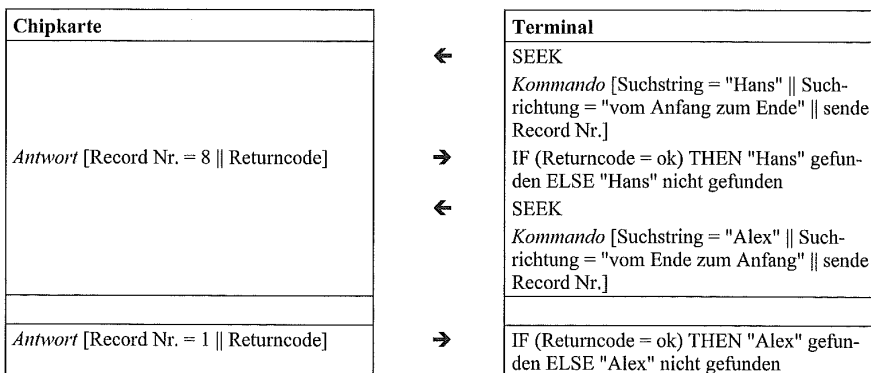


Bild 7.11 Beispiel für den Kommandoablauf mit einem SEEK-Kommando.

Die ISO/IEC 7816-9 beschreibt zwei Kommandos zur Suche von Daten in transparenten und Record-orientierten Dateien. Das SEARCH RECORD-Kommando ist die Variante von ISO/IEC des GSM 11.11 Kommandos SEEK. Der große Unterschied zwischen den beiden ist, dass nach ISO/IEC 7816-9 noch ein Short-FID zur impliziten Selektion eines EFs innerhalb des Suchkommandos mit übergeben werden kann.

Tabelle 7.15 Die Funktionalität von SEARCH RECORD nach ISO/IEC 7816-9

SEARCH RECORD	
Kommando	• Länge des Suchstrings • Suchstring • Offset • Modus (vom Anfang an vorwärts, vom Ende aus rückwärts, ab der nächsten Position vorwärts, ab der vorherigen Position rückwärts) • <i>Schalter</i>: Recordnummer des gefundenen Records zurücksenden
	• <i>optional</i>: Short-FID für implizite Selektion
Antwort	• Recordnummer (falls via Schalter gewählt) • Returncode

Mit SEARCH BINARY ist es möglich, in einer vorab explizit oder im Kommando implizit selektierten Datei der Struktur „transparent“ nach zu übergebenden Daten zu suchen. Als Ergebnis erhält man den Offset, gerechnet vom Anfang der Datei, zum ersten Byte des gefundenen Suchstrings.

Tabelle 7.16 Die Funktionalität von SEARCH BINARY nach ISO/IEC 7816-9

SEARCH BINARY	
Kommando	• Länge des Suchstrings • Suchstring • Offset • <i>optional</i>: Short-FID für implizite Selektion
Antwort	• Offset zu den gefundenen Daten • Returncode

7.4 Operationen auf Dateien

Es existiert eine Reihe von Kommandos, mit denen Dateiinhalte nicht nur durch bloßes Schreiben verändert werden können. Die beiden Hauptvertreter dieser Klasse sind die Kommandos INCREASE und DECREASE. Sie erhöhen bzw. verringern den als Zähler aufgebauten Dateiinhalt mit zyklischer Struktur um einen zu übergebenden Wert.

Der Grund für die vorgeschriebene zyklische Dateistruktur ist die so gegebene Protokollfunktion. Diese beiden Kommandos werden vor allem für einfache Kleingeldbörsen oder Zähler benutzt.

Tabelle 7.17 Die Funktionalität von DECREASE nach GSM 11.11

DECREASE	
Kommando	• zu subtrahierender Wert
Antwort	• subtrahierter Wert • neuer Wert des Records • Returncode

Tabelle 7.18 Die Funktionalität von INCREASE nach GSM 11.11

INCREASE	
Kommando	• zu addierender Wert
Antwort	• addierter Wert • neuer Wert des Records • Returncode

Für das folgende Beispiel sei der Einfachheit halber eine zyklische Datei mit nur einem Record angenommen. Der Anfangswert des Records sei 10. Nach Beendigung des Ablaufs hat der Inhalt wieder denselben Wert.

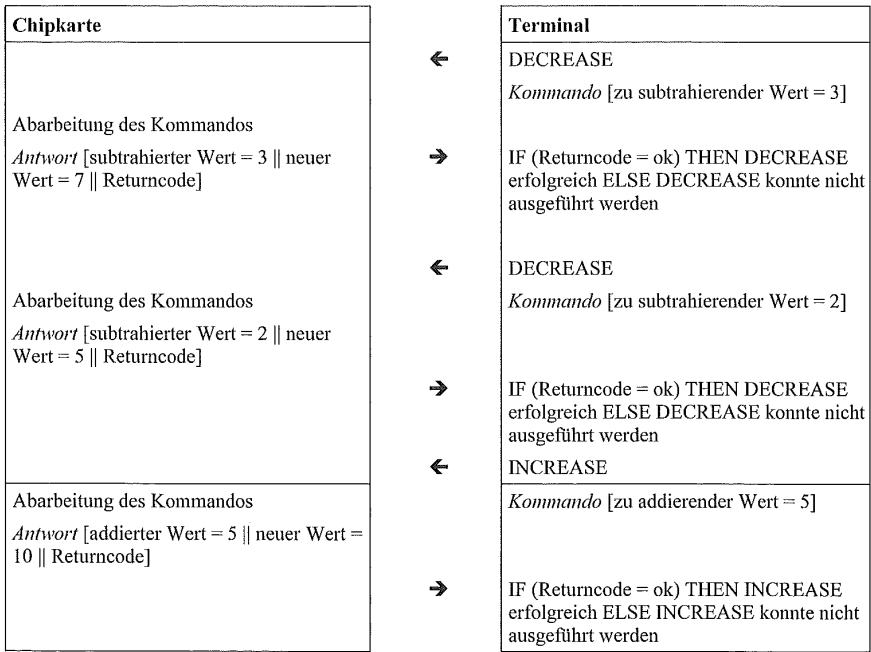


Bild 7.12 Beispiel für einen Kommandoablauf mit INCREASE und DECREASE.

Ein Kommando, das gewissermaßen auch eine Operation auf einem EF ausführt, ist EXECUTE. Mit ihm können ablauffähige Programme in EFs mit der Struktur „executable“ gestartet werden. Das auszuführende Programm kann sowohl im Kommando Daten vom Terminal erhalten als auch erzeugte Daten in der Antwort an das Terminal zurücksenden.

Das Kommando selber bzw. die dazugehörige Dateistruktur ist umstritten, da damit unter Umständen das gesamte Sicherheitssystem einer Chipkarte umgangen werden kann.

Tabelle 7.19 Die Funktionalität von EXECUTE nach EN 726-3

EXECUTE	
Kommando	• der executable-Datei zu übergebende Daten
Antwort	• von der executable-Datei zurückerhaltene Daten • Returncode

7.5 Identifizierungskommandos

Neben der Verwendung von Chipkarten als geschützte Datenträger nutzt man sie auch zur Identifizierung von Personen. Das dazu übliche Verfahren ist die Identifizierung durch Übergabe eines nur der Chipkarte und dem Benutzer bekannten Geheimnisses. Dies ist in der Regel eine PIN (*personal identification number*), umgangssprachlich auch Geheimzahl genannt.

Ein Verfahren, das jedermann aus persönlicher Erfahrung kennt, ist die Überprüfung der PIN. Man gibt an einem Kartenterminal die PIN ein und bekommt kurz darauf auf einem Display eine Anzeige, ob die PIN korrekt war bzw. wie viele Fehlversuche noch möglich sind. Dabei erhält die Chipkarte vom Terminal in dem Kommando VERIFY eine üblicherweise 4-stellige PIN und vergleicht diese mit der im EEPROM abgespeicherten. Ist die von außen übergebene und die gespeicherte PIN gleich, so beeinflusst dies den Zustandsautomaten in der Chipkarte, und das Terminal erhält eine Meldung über den positiven Vergleich. Der Fehlbedienungsähler der PIN erhält daraufhin ebenfalls seinen Ursprungswert (d. h. 0) zurück. Stimmt die übergebene PIN mit der in der Karte gespeicherten nicht überein, so bewirkt dies eine Erhöhung des Fehlbedienungsählers. Ist dieser bei dem definierten Maximalwert angelangt, so ist die Karte für jede weitere PIN-Überprüfung gesperrt.

In vielen Chipkarten-Betriebssystemen ist es möglich, mehrere PINs zu verwenden. Dann ist es unabdingbar, bei allen die PIN betreffenden Kommandos die entsprechende Nummer der PIN zu übergeben, damit diese auch korrekt adressiert werden kann. In der Regel legen Kartenherausgeber aber großen Wert darauf, dass nur eine einzige PIN pro Karte existiert, auch wenn technisch mehrere möglich wären. Dies ist aus Gründen der Kundenakzeptanz und Benutzerfreundlichkeit unabdingbar.

Die Abkürzung PIN nennt man im Bereich der Telekommunikation oft auch CHV, was die Kurzform von Chip Holder Verification ist und genau das Gleiche wie PIN bedeutet. Da manche der beschriebenen Kommandos aus diesem Bereich stammen, wird im Kommandonamen das Kürzel CHV statt PIN verwendet.

Tabelle 7.20 Die Funktionalität von VERIFY CHV nach GSM 11.11

VERIFY CHV	
Kommando	• PIN • Nummer der PIN
Antwort	• Returncode

Die ISO/IEC 7816-4 beschreibt ein mit GSM 11.11 weitgehend identisches Kommando zur Prüfung der PIN. Es hat den Namen VERIFY und könnte nicht nur für einen PIN-Vergleich sondern auch für die Überprüfung von biometrischen Merkmalen benutzt werden. Gegenüber der PIN-Prüfung nach GSM besteht in der Codierung nur ein signifikanter Unterschied. Die ISO/IEC sieht im Kommando eine Unterscheidung zwischen einer globalen PIN und einer applikationsspezifischen PIN vor. Damit ist es möglich, im Kommando auszuwählen, ob die für die gesamte Chipkarte geltende PIN oder eine nur im aktuellen DF gültige PIN geprüft werden soll.

Tabelle 7.21 Die Funktionalität von VERIFY nach ISO/IEC 7816-4.

VERIFY	
Kommando	• PIN oder biometrisches Merkmal (= Geheimnis) • Nummer der PIN oder des biometrischen Merkmal • <i>Schalter</i>: globales oder applikationsspezifisches Geheimnis
Antwort	• Returncode

Bei manchen Anwendungen soll der Kartenbenutzer bei der ersten PIN-Eingabe die ihm am geeignetsten erscheinende PIN selber wählen. Man kann hierzu das so genannte Null-pin-Verfahren anwenden. Dazu wird die PIN bei der Personalisierung mit Nullen vorbelegt, was für manche Chipkarten-Betriebssysteme eine besondere Bedeutung hat. Das Betriebssystem weist alle VERIFY-Kommandos mit Null-PIN ab und fordert eine vorherige Änderung der PIN mit CHANGE REFERENCE DATA. Damit wird der Benutzer gezwungen, seine PIN zu ändern, da er mit der Eingabe einer aus lauter Nullen bestehenden PIN in der Chipkarte keinen Sicherheitszustand verändern kann. Dieses Verfahren ist nicht standardisiert, lässt sich jedoch in bestimmten Fällen, und wenn es vom Betriebssystem unterstützt wird, manchmal nutzbringend einsetzen.

Seit der Einführung von PINs als Identifizierungsmerkmal für Kartenbenutzer nimmt deren Anzahl immer mehr zu. Ein durchschnittlicher Kartenbenutzer müsste sich heute wahrscheinlich schon 10 oder 20 verschiedene PINs für seine diversen Karten und andere Zugriffsberechtigungen merken. Dass dies unrealistisch ist, sieht man am Beispiel der vielen Personen, die ihre PIN auf die Karte schreiben. Die Verwendung von Chipkarten ermöglicht es dem Benutzer aber nun, seine PIN frei auszuwählen und eine einheitliche PIN für alle seine Karten zu verwenden. Dies mag zwar sicherheitstechnisch problematisch sein, denn wenn ein Angreifer eine PIN ausspähen kann, kennt er daraufhin alle, doch ist es immer noch besser, als die PIN für jeden sichtbar außen auf die Karte zu notieren.

Das Kommando CHANGE CHV stellt das Mittel zur Änderung der PIN dar. Das ISO/IEC 7816-8-Äquivalent dieses Kommandos heißt CHANGE REFERENCE DATA und besitzt ansonsten gleiche Ein- und Ausgangsparameter. Mit dem Wissen über die ak-

tuell in der Karte enthaltene PIN ist es somit möglich, sie durch eine neue PIN zu ersetzen. Sollte die aktuelle PIN nicht korrekt sein, so inkrementiert das Betriebssystem den Fehlbedienungs­zähler. Damit wehrt man einen denkbaren Angriff über dieses Kommando auf die PIN ab. Sobald die aktuelle PIN korrekt an die Chipkarte übergeben ist, trägt diese die übergebene neue PIN in den entsprechenden Speicher ein und setzt den Fehlbedienungs­zähler wieder auf seinen Ursprungswert.

Tabelle 7.22 Die Funktionalität von CHANGE CHV nach GSM 11.11

CHANGE CHV	
Kommando	• alte PIN • neue PIN • Nummer der PIN
Antwort	• Returncode

Ist der Fehlbedienungs­zähler einer PIN abgelaufen, kann man ihn mit dem Kommando UNBLOCK CHV und einer zweiten PIN, die PUK (*personal unblocking key*) genannt wird, wieder auf seinen Ursprungszustand setzen. Diese PUK hat üblicherweise eine größere Länge (z. B. 8-stellig) als die normale 4-stellige PIN. Der Benutzer muss sich die PUK auch nicht dauernd merken, da er diese nur im Fall einer vergessenen PIN kennen muss. Es genügt dann, wenn er gegebenenfalls zuhause Unterlagen über seine PUK findet. Es nutzt jedoch dem Benutzer sehr wenig, wenn er den Fehlbedienungs­zähler seiner vergessenen PIN auf seinen Ursprungswert zurücksetzt, da er damit seine vergessene PIN auch nicht mehr herausfinden kann. Deshalb muss mit UNBLOCK CHV eine neue PIN an die Chipkarte übergeben werden.

Bei Hybridkarten, die sowohl Chip als auch Magnetstreifen enthalten, darf es mit diesem Kommando nicht möglich sein, eine neue PIN an den Chip zu übergeben. Denn dann wäre auf dem Magnetstreifen eine andere PIN als im Chip enthalten, was große Probleme hervorrufen würde. Deshalb setzt man in diesem Fall den Fehlbedienungs­zähler nur auf seinen Ursprungswert und schickt dem Kunden einen Brief mit seiner alten PIN.

Tabelle 7.23 Die Funktionalität von UNBLOCK CHV nach GSM 11.11

UNBLOCK CHV	
Kommando	• PUK • neue PIN
Antwort	• Returncode

Die ISO/IEC 7816-8 sieht ein eigenes Kommando vor, mit dem Fehlbedienungs­zähler, die ihren Maximalwert erreicht haben, wieder auf den Ausgangswert zurückgesetzt werden können. Die Bezeichnung des Kommandos ist RESET RETRY COUNTER. Um dieses Kommando ausführen zu dürfen, muss ein bestimmter Sicherheitszustand erfüllt sein, der in der Regel eine erfolgreiche Authentisierung ist. Im Gegensatz zu seinem Namen lässt sich dieses Kommando auch zum Ersetzen der aktuellen PIN durch eine neue PIN benutzen.

Tabelle 7.24 Die Funktionalität von RESET RETRY COUNTER ISO/IEC 7816-8

RESET RETRY COUNTER	
Kommando	• Nummer der PIN • <i>Option:</i> PUK neue PIN • <i>falls Option gewählt:</i> Nummer der PUK • <i>Schalter:</i> globale PIN oder applikationsspezifische PIN
Antwort	• Returncode

Bei GSM gibt es noch zwei weitere Kommandos, durch die die Abfrage der PIN beeinflusst werden kann. Es sind dies DISABLE CHV und ENABLE CHV, die zum Ab- und Anschalten der PIN-Prüfung dienen. Ist die PIN-Prüfung abgeschaltet, so sind alle Zugriffsbedingungen auf Dateien, die eine vorherige PIN-Prüfung erfordern, außer Kraft gesetzt. Beide Kommandos erfreuen sich großer Beliebtheit im Mobilfunkbereich, da nicht nach jedem Anschalten des Mobiltelefons die PIN neu eingegeben werden muss. Sicherheitstechnisch sind beide Kommandos nicht unbedenklich, da der Schutz, den eine PIN vor unbefugter Benutzung bietet, damit ausgeschaltet werden kann. Allerdings kann durch den Benutzer auch eine Trivial-PIN, wie "0000", mit CHANGE CHV gewählt werden, deren Sicherheit ebenfalls sehr niedrig ist.

Die beiden ISO/IEC 7816-8 Kommandos für diese Funktionalität haben die Namen ENABLE VERIFICATION REQUIREMENT und DISABLE VERIFICATION REQUIREMENT. Je nach Anwendung muss vorab ein bestimmter Sicherheitszustand erreicht sein, damit eines dieser Kommandos ausgeführt werden darf.

Tabelle 7.25 Die Funktionalität von ENABLE VERIFICATION REQUIREMENT nach ISO/IEC 7816-8

ENABLE VERIFICATION REQUIREMENT	
Kommando	• Referenzdaten (z. B. PIN) • Nummer der Referenzdaten
Antwort	• Returncode

Tabelle 7.26 Die Funktionalität von DISABLE VERIFICATION REQUIREMENT nach ISO/IEC 7816-8

DISABLE VERIFICATION REQUIREMENT	
Kommando	• Referenzdaten (z. B. PIN) • Nummer der Referenzdaten
Antwort	• Returncode

Die obigen Verfahren zur PIN-Überprüfung sind aus einleuchtenden Gründen Angriffen ausgesetzt, da sich mit einer gefundenen oder gestohlenen Karte und der richtigen PIN unter Umständen ein großer geldwerter Vorteil erreichen lässt. Alle diese Kommandos, die

im Zusammenhang mit dem PIN- oder PUK-Vergleich stehen, müssen gegenüber Analysen des elektrischen Verhaltens oder des Zeitverhaltens der Chipkarte abgesichert sein. Beispielsweise darf der Stromverbrauch bei VERIFY in Abhängigkeit von korrekter oder falscher PIN nicht unterschiedlich sein. Ebenso darf auch das Zeitverhalten bei Ausführung der PIN-Kommandos keine Abhängigkeit von echter oder falscher PIN haben. Unterschiedliche Laufzeiten können sich fatal auf die Sicherheit der Karte und damit letztendlich auf das gesamte System auswirken. Denn dann könnte man auf sehr einfachem Weg die richtige PIN herausfinden, und alle PINs des Systems wären als Identifizierung der Benutzer hinfällig.

7.6 Authentisierungskommandos

Zusätzlich zur Identifizierung des Kartenbesitzers existieren noch eine Reihe von Kommandos zur Authentisierung von Terminal und Karte. Da jeder dieser beiden Kommunikationspartner mit einem vollständigen Computer ausgerüstet ist, können die verwendeten Abläufe gegenüber den PIN-Prüfungen wesentlich aufwendiger und somit auch sicherer gemacht werden.

Bei der PIN-Prüfung erhält die Karte ein Geheimnis im Klartext (die PIN) über die Schnittstelle und muss dieses nur mit einer im Speicher enthaltenen PIN vergleichen. Ein Abhören der Übertragungsleitung hätte demnach fatale Folgen. Moderne Authentisierungsverfahren sind so aufgebaut, dass dieser Angriff unmöglich gemacht wird.

Prinzipiell wird bei der Authentisierung ein gemeinsames Geheimnis beider Kommunikationspartner überprüft, ohne dass dieses über die Schnittstelle gesendet werden muss. Die Verfahren sind so aufgebaut, dass ein Abhören der Datenübertragung die Sicherheit der Authentisierung nicht beeinträchtigt.²

Je nach Betriebssystem existieren verschiedene Kommandos zur Authentisierung von Karte oder Terminal oder beiden gleichzeitig. Der Anschaulichkeit halber ist hier und im folgenden Teil des Kapitels immer nur von Authentisierungen zwischen Karte und Terminal die Rede. Informationstechnisch gesehen, authentisiert sich aber der „Rest der Welt“ gegenüber einer Anwendung in der Karte. Es wird dabei auch nicht überprüft, ob die Karte als Ganzes echt ist, sondern lediglich, ob der implantierte Mikrocontroller ein gemeinsames Geheimnis mit der äußeren Welt besitzt. Dies muss man bei manchen Anwendungen bedenken.

In vielen Betriebssystemen sind die für die Authentisierung verwendeten Schlüssel durch einen Fehlbedienungszähler (*retry counter*) geschützt. Versucht nun ein Terminal zu oft eine negativ verlaufende Authentisierung, dann sperrt die Chipkarte den betreffenden Schlüssel für weitere Authentisierungen. Dies ist vom Standpunkt der Systemsicherheit ein einwandfreies Vorgehen, es hat aber einen Nachteil: Die Fehlbedienungszähler der Authentisierungsschlüssel wieder auf die Ausgangswerte zurückzusetzen, führt oft zu sehr aufwendigen und logistisch schwierigen und teuren Administrationsprozeduren. Deshalb haben in manchen Systemen die Authentisierungsschlüssel keine Fehlbedienungszähler.

Zur Authentisierung sollten aus Sicherheitsgründen nur kartenindividuelle Schlüssel verwendet werden. Die Erzeugung dieser Schlüssel kann auf der Grundlage eines einzigartigen

² Siehe auch Abschnitt 4.11 „Authentisierung“.

Merkmals der Chipkarte erfolgen. Eine Seriennummer oder Chip-Fabrikationsnummer eignet sich dafür sehr gut. Diese nicht geheime und damit öffentliche Nummer kann mit einem Kommando aus der Karte gelesen werden. Dazu gibt es momentan noch keine Norm, das Kommando ist hier als GET CHIP NUMBER aufgeführt. Dieser Name ist aber von Betriebssystem zu Betriebssystem, genauso wie die ausgetauschten Daten, mit Sicherheit unterschiedlich. In diesem Zusammenhang ist jedoch nur die Funktionalität entscheidend. GET CHIP NUMBER holt also von der Chipkarte eine in Hinblick auf den DES-Algorithmus vorzugsweise 8 Byte lange einzigartige Seriennummer ab, die zur eindeutigen Identifizierung des Chips und zur Berechnung von kartenindividuellen Schlüsseln verwendet wird.

Tabelle 7.27 Die Funktionalität von GET CHIP NUMBER

GET CHIP NUMBER	
Kommando	• ---
Antwort	• Chipnummer • Returncode

Es ist noch ein weiteres zusätzliches Kommando für eine Authentisierung notwendig: GET CHALLENGE nach ISO/IEC 7816-4. Mit ihm fordert man von der Chipkarte eine Zufallszahl an, welche für die spätere Authentisierung benötigt wird. Die Länge beträgt bei einer Authentisierung mit DES typischerweise 8 Byte, kann jedoch bei einem anderen kryptografischen Algorithmus unterschiedlich sein.

Um die folgenden Ablaufbeispiele übersichtlich zu gestalten und nicht unnötig zu verkomplizieren, wurde die Ableitung des kartenindividuellen Schlüssels nicht ausgeführt. Sie ist aber aus Sicherheitsgründen zwingend notwendig.

Tabelle 7.28 Die Funktionalität von GET CHALLENGE nach ISO/IEC 7816-4

ASK RANDOM / GET CHALLENGE	
Kommando	• ---
Antwort	• Zufallszahl • Returncode

Das Kommando INTERNAL AUTHENTICATE fungiert zur Authentisierung der Karte oder, bei Multiapplication-Chipkarten, einer Anwendung durch das Terminal. Man kann mit diesem Kommando sicherstellen, dass die Karte echt ist. Dazu erhält die Chipkarte eine Zufallszahl, die sie mit einem nur ihr und dem Terminal bekannten Schlüssel und einem kryptografischen Algorithmus (z. B. DES) verschlüsselt. Das Ergebnis dieser Verschlüsselungsoperation erhält das Terminal in der Antwort zurück. Es führt nun seinerseits die gleiche Verschlüsselung wie in der Chipkarte durch und vergleicht das Ergebnis mit dem von der Chipkarte erhaltenen. Sind die beiden gleich, dann kennt auch die Chipkarte den geheimen Authentisierungsschlüssel, und er muss daher echt sein. Die Chipkarte ist nun authentisiert.

Insofern ist mit diesem Kommando das klassische Challenge-Response-Verfahren zur Authentisierung eines Kommunikationspartners realisiert. Der genaue Dateninhalt der

Challenge ist dabei nach ISO/IEC 7816-4 nicht exakt festgelegt. Dort ist nur genormt, dass es sich um einen zufälligen und sitzungsspezifischen Wert handelt, der an die Chipkarte gesendet wird. Dies hat zur Folge, dass bei INTERNAL AUTHENTICATE und auch den im Folgenden beschriebenen Authentisierungskommandos immer eine detaillierte Spezifikation erforderlich ist, damit sie interoperabel eingesetzt werden können.

Tabelle 7.29 Die Funktionalität von INTERNAL AUTHENTICATE nach IEO/IEC 7816-4

INTERNAL AUTHENTICATE	
Kommando	• Zufallszahl • Nummer des zu verwendenden Algorithmus • Nummer des zu verwendenden Schlüssels
Antwort	• enc (Schlüssel; Zufallszahl) • Returncode

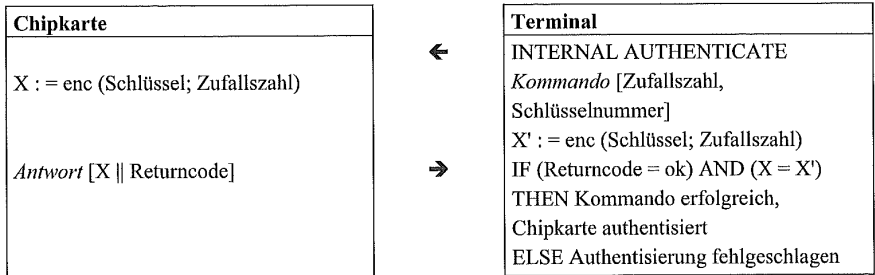


Bild 7.13 Beispiel für einen Kommandoablauf bei INTERNAL AUTHENTICATE.

Das Kommando EXTERNAL AUTHENTICATE verwendet das Terminal, um der Chipkarte zu zeigen, dass sie Kontakt mit einem echten Terminal hat. Angestoßen werden muss das Kommando durch das Terminal, da der festgelegte Kommunikationsablauf immer im Rahmen des Challenge-Response-Verfahrens ablaufen muss. Jedoch kann die Chipkarte eine Authentisierung des Terminals erzwingen, indem sie den Zugriff auf bestimmte Dateien so lange sperrt, bis sich das Terminal erfolgreich authentisiert hat.

Tabelle 7.30 Die Funktionalität von EXTERNAL AUTHENTICATE nach ISO/IEC 7816-4

EXTERNAL AUTHENTICATE	
Kommando	• enc (Schlüssel; Zufallszahl) • Nummer des zu verwendenden Algorithmus • Nummer des zu verwendenden Schlüssels
Antwort	• Returncode

Der Ablauf ist dabei Folgender: Als Erstes fordert das Terminal mit dem Kommando GET CHALLENGE eine Zufallszahl von der Chipkarte an, die es dann mit einem geheimen Schlüssel verschlüsselt. In dem zweiten Kommando, EXTERNAL AUTHENTICATE schickt das Terminal die verschlüsselte Zufallszahl wieder zur Chipkarte zurück. Diese führt die gleiche Verschlüsselungsoperation mit dem ihr ebenfalls bekannten geheimen

Schlüssel aus und vergleicht das errechnete Ergebnis mit dem vom Terminal erhaltenen. Sind beide gleich, muss das Terminal ebenfalls im Besitz des geheimen Authentisierungsschlüssels sein und ist somit authentisiert.

Nach positiver Authentisierung des Terminals ändert das Betriebssystem den Zustand im Zustandsautomaten der Karte. Das Terminal darf nun auf bestimmte Dateien schreibend oder lesend zugreifen. Dies ist dann die für den Benutzer erkennbare Auswirkung einer externen Authentisierung.

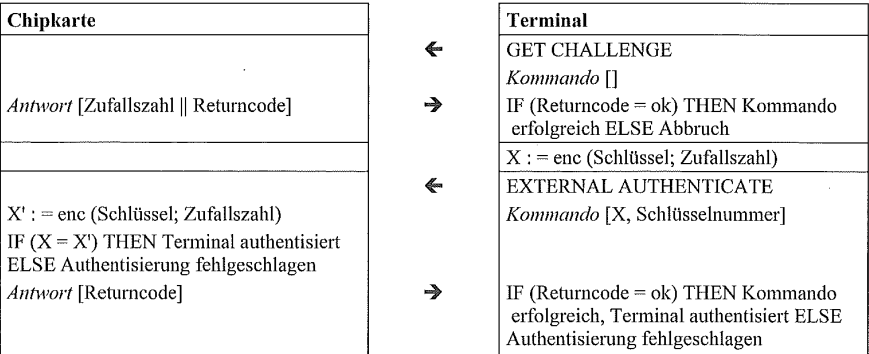


Bild 7.14 Beispiel für einen Kommandoablauf bei EXTERNAL AUTHENTICATE.

Führt man INTERNAL und EXTERNAL AUTHENTICATE nacheinander aus, sind beide Kommunikationsteilnehmer gegenseitig authentisiert. Beide wissen also, dass der jeweils andere echt ist. Dazu sind aber insgesamt drei komplette Kommandosequenzen notwendig. Um dieses aufwendige und zeitintensive Verfahren zu vereinfachen, wurden die drei Kommandos mit ihren Übertragungsdaten miteinander verknüpft und die beiden Authentisierungskommandos zu einem einzigen zusammengefasst.

Der Name dieses Kommandos ist MUTUAL AUTHENTICATE, und es ist in ISO/IEC 7816-8 festgelegt. Mit ihm kann eine gegenseitige Authentisierung von zwei Instanzen nach ISO/IEC 9798-2 und -3 realisiert werden. Durch dieses einzelne Authentisierungskommando erhöht sich auch die Sicherheit der ganzen Prozedur, da es nun nicht mehr möglich ist, in unlauterer Absicht Kommandos zwischen die zwei einseitigen Authentisierungen einzufügen. Eine weitere Erhöhung der Sicherheit des Verfahrens beruht darauf, dass es nicht möglich ist, durch Abhören der Kommunikation zwischen Terminal und Chipkarte Klartext-Schlüsseltext-Paare zu erhalten, welche eine ideale Grundlage für einen Angriff sind.

Tabelle 7.31 Die Funktionalität von MUTUAL AUTHENTICATE nach ISO/IEC 7816-8

MUTUAL AUTHENTICATE	
Kommando	• enc (Schlüssel; Zufallszahl_Terminal, Zufallszahl_Chipkarte, Chipnummer) • Nummer des zu verwendenden Algorithmus • Nummer des zu verwendenden Schlüssels
Antwort	• enc (Schlüssel; Zufallszahl_Chipkarte, Zufallszahl_Terminal) • Returncode

Der Funktionsablauf der gegenseitigen Authentisierung ist folgendermaßen aufgebaut: Als Erstes fordert das Terminal mit GET CHIP NUMBER die Chipnummer der Chipkarte an. Nun kann das Terminal die kartenindividuellen Schlüssel berechnen. Anschließend erhält das Terminal mit GET CHALLENGE eine Zufallszahl von der Karte und erzeugt daraufhin seinerseits eine Zufallszahl.

Nachdem das Terminal die Zufallszahl der Chipkarte erhalten hat, bildet es aus den beiden Zufallszahlen und der Chipnummer einen Block, den es mit dem geheimen Authentisierungsschlüssel und einem kryptografischen Algorithmus im CBC-Modus verschlüsselt. Den erhaltenen Schlüsseltextblock sendet es zur Chipkarte, welche den Block wieder entschlüsselt und Chipnummer sowie die Zufallszahl mit der vorher abgesendeten vergleicht. Sind die beiden gleich, dann ist das Terminal authentisiert.

Nun vertauscht die Chipkarte die beiden Zufallszahlen, lässt die Chipnummer weg und verschlüsselt das Ganze wieder mit dem geheimen Authentisierungsschlüssel. Nachdem das Terminal diesen Block erhalten und entschlüsselt hat, kann es durch Vergleich der bekannten Zufallszahlen feststellen, ob die Karte im Besitz des geheimen Authentisierungsschlüssels war. Ist dies der Fall, so ist auch die Chipkarte authentisiert.

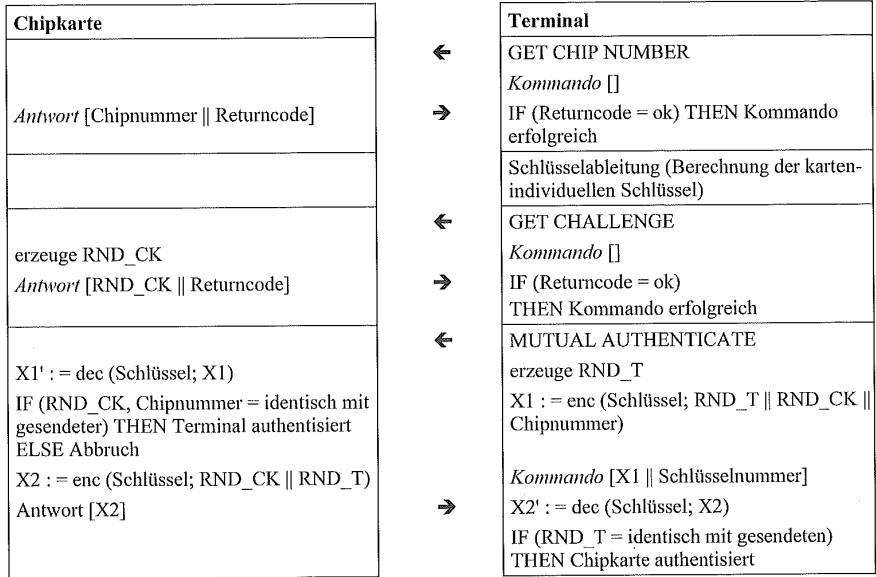


Bild 7.15 Beispiel für einen Kommandoablauf bei MUTUAL AUTHENTICATE.

7.7 Kommandos für kryptografische Algorithmen

Kommandos für kryptografische Algorithmen sind für viele Anwendungen von großer Bedeutung, da mit ihnen beispielsweise Chipkarten ohne großen Aufwand als Ver- und Entschlüsselungseinrichtungen oder für digitale Signaturen benutzt werden können. Viele Chipkarten-Betriebssysteme bieten eigene Befehlssätze für die Ausführung von kryptogra-

fischen Funktionen an. Chipkarten-Kommandos wie ENCRYPT, DECRYPT, SIGN DATA und VERIFY SIGNATURE sind entstanden, weil es lange Zeit keine Norm für diese Art von Funktionalität gab. In der ISO/IEC 7816-8 sind jedoch mittlerweile Kommandos festgelegt, die speziell für die Abarbeitung von kryptografischen Algorithmen in Chipkarten ausgelegt sind.

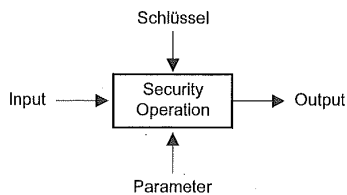


Bild 7.16 Das grundlegende Prinzip der beiden ISO/IEC 7816-8 Kommandos für die Ausführung von kryptografischen Funktionen MANAGE SECURITY ENVIRONMENT und PERFORM SECURITY OPERATION.

Die Funktionalität von kryptografisch relevanten Funktionen ist nach ISO/IEC 7816-8 auf zwei Kommandos verteilt. Mit dem Kommando MANAGE SECURITY ENVIRONMENT (MSE) können vor der eigentlichen Ausführung des kryptografischen Algorithmus verschiedene Rahmenparameter dazu eingestellt werden. Mit dem Kommando wird der Karte ein so genanntes Template übergeben, das die entsprechenden Parameter enthält. Diese sind dann so lange unveränderlich gültig, bis sie mit einem erneuten MANAGE SECURITY ENVIRONMENT wieder neu gesetzt werden. Die Templates selber bestehen aus TLV-codierten Datenobjekten, was eine maximale Variabilität - und leider auch Komplexität - bei der Parameterübergabe zulässt.

Tabelle 7.32 Die Funktionalität von MANAGE SECURITY ENVIRONMENT nach ISO/IEC 7816-8.

MANAGE SECURITY ENVIRONMENT	
Kommando	• Template mit Parametern für kryptografische Prüfsumme oder Template mit Parametern für digitale Signatur oder Template mit Parametern für Hash-Algorithmus oder Template mit Parametern für Authentisierung oder Template mit Parametern für Ver- und Entschlüsselung
Antwort	• Returncode

Nachdem mit MANAGE SECURITY ENVIRONMENT alle Optionen für die kryptografische Funktion gesetzt sind, kann das Kommando PERFORM SECURITY OPERATION (PSO) aufgerufen werden. Mit diesem lassen sich nun die unterschiedlichsten Sicherheitsoperationen ausführen, vorausgesetzt dass diese vom Chipkarten-Betriebssystem unterstützt werden. Die Anzahl der Möglichkeiten bei diesem Kommando ist jedoch so groß, dass dies nicht immer zwangsläufig der Fall sein muss. Obwohl PERFORM SECUR-

RITY OPERATION nur durch ein einziges Instruction-Byte codiert ist, hat es acht grundlegend unterschiedliche Funktionen, die über das Parameter Byte P1 unterschieden wird. Der Grund dafür war, dass mittlerweile die Anzahl der zur Codierung von Kommandos zur Verfügung stehenden Instruction-Bytes knapp werden. Aufgrund dieser vielfältigen Möglichkeiten, welche PERFORM SECURITY OPERATION bietet, ist in den folgenden Absätzen ebenfalls eine vergleichbare Auftrennung erfolgt, ohne dass jeweils im Einzelnen darauf hingewiesen wird.

PERFORM SECURITY OPERATION, mit der Option COMPUTE CRYPTOGRAPHIC CHECKSUM ausgeführt, wird zur Berechnung einer kryptografischen Prüfsumme CCS (*cryptographic checksum*), vulgo MAC, benutzt. Das zu benutzende Padding sowie die verwendeten Schlüssel sind entweder bereits implizit durch das Betriebssystem vorgegeben oder müssen mit MANAGE SECURITY ENVIRONMENT explizit angegeben werden. Das Pendant zu dieser Kommando-Option ist VERIFY CRYPTOGRAPHIC CHECKSUM, das für übergebene Daten eine kryptografische Prüfsumme berechnet und diese mit einer ebenfalls übermittelten Referenz vergleicht. Das Ergebnis ist eine anschließend an das Terminal zurückgesandte Gleich-Ungleich-Aussage.

Tabelle 7.33 Die Funktionalität von PERFORM SECURITY OPERATION mit der Option COMPUTE CRYPTOGRAPHIC CHECKSUM nach ISO/IEC 7816-8

PERFORM SECURITY OPERATION	
Option: COMPUTE CRYPTOGRAPHIC CHECKSUM	
Kommando	• zu verschlüsselnde Daten
Antwort	• CCS • Returncode

Tabelle 7.34 Die Funktionalität von PERFORM SECURITY OPERATION mit der Option VERIFY CRYPTOGRAPHIC CHECKSUM nach ISO/IEC 7816-8

PERFORM SECURITY OPERATION	
Option: VERIFY CRYPTOGRAPHIC CHECKSUM	
Kommando	• zu verschlüsselnde Daten • CCS
Antwort	• Returncode

Für die reine Ver- und Entschlüsselung von Daten sind die beiden im Folgenden vorgestellten Optionen des Kommandos PERFORM SECURITY OPERATION vorgesehen. Die Funktionalität der Kommandooption ENCIPHER ist die Verschlüsselung von übergebenen Daten. Der dabei verwendete Verschlüsselungsalgorithmus kann je nach Betriebssystem vorab mit MANAGE SECURITY ENVIRONMENT gewählt werden. Desgleichen muss auch der Modus des Verschlüsselungsalgorithmus über vorab zu übergebende Parameter festgelegt worden sein. Bei einem Blockverschlüsselungsalgorithmus könnte man also zwischen ECB- und CBC-Modus auswählen. Da die Länge der an die Chipkarte gesendeten Daten nicht unbedingt ein Vielfaches der Blocklänge des kryptografischen Algorith-

mus sein muss, ist es notwendig, mit einem weiteren Parameter die Padding-Methode festzulegen. Ebenfalls wichtig ist die Adressierung eines in der Chipkarte enthaltenen Schlüssels, den der Algorithmus zur Verschlüsselung benötigt.

Tabelle 7.35 Die Funktionalität von `PERFORM SECURITY OPERATION` mit der Option `ENCIPHER` nach ISO/IEC 7816-8

PERFORM SECURITY OPERATION	
Option: ENCIPHER	
Kommando	• zu verschlüsselnde Daten
Antwort	• verschlüsselte Daten • Returncode

Die Umkehrfunktion zur `ENCIPHER`-Option heißt `DECIPHER`. Damit können in dem gleichen Modus wie bei `ENCIPHER` die übergebenen Daten wieder entschlüsselt werden. Dazu ist in der Chipkarte natürlich die Kenntnis des entsprechenden Schlüssels, des Modus und der Padding-Methode notwendig. Dies muss dem Betriebssystem vorab mit `MANAGE SECURITY ENVIRONMENT` mitgeteilt werden.

Tabelle 7.36 Die Funktionalität von `PERFORM SECURITY OPERATION` mit der Option `DECIPHER` nach ISO/IEC 7816-8

PERFORM SECURITY OPERATION	
Option: DECIPHER	
Kommando	• verschlüsselte Daten
Antwort	• entschlüsselte Daten • Returncode

Mit dem Einzug von Public-Key-Algorithmen in den Chipkartenbereich entstand auch die Notwendigkeit, mit entsprechenden Kommandos die neu hinzugekommenen Funktionen zu nutzen. Gerade Chipkarten eignen sich hervorragend für digitale Signaturen, da der geheime Schlüssel für den Signaturalgorithmus im Speicher sicher und nicht auslesbar aufbewahrt werden kann. Die ISO/IEC 7816-8 beschreibt vier Kommandooptionen, die vor allem im Umfeld von digitaler Signatur eingesetzt werden können.³

Mit der Option `HASH` des Kommandos `PERFORM SECURITY OPERATION` ist es möglich, einen Hash-Wert zu berechnen. Dazu können dem Kommando entweder alle zu hashenden Daten übergeben werden oder ein bereits außerhalb der Chipkarte vorab berechneter Hash-Wert mit den dazugehörigen Daten, welche für den letzten Berechnungsschritt notwendig sind. Die Hash-Berechnung für den letzten Block kann dann abschließend in der Chipkarte vollzogen werden. Der Vorteil dieses Vorgehens besteht darin, dass die Hashwertbildung außerhalb der Chipkarte wesentlich schneller vonstatten geht, aber trotzdem noch der letzte Schritt in der Chipkarte durchgeführt wird. Es bietet aus rein kryptologischer Sicht wenig zusätzliche Sicherheit, schränkt aber den Spielraum für Manipulationen des Hashwertes doch etwas ein, weshalb es in der Praxis durchaus verwendet wird.

³ Siehe auch Abschnitt 14.4 „Digitale Signatur“.

Da die zu hashenden Daten in der Regel größer sind als die maximal zulässige Länge des Datenteils eines Kommandos, besitzt HASH ein so genanntes Schicht-7-Chaining, was bedeutet, dass auf der Anwendungsschicht Datenblöcke logisch miteinander verkettet werden. Der Letzte zu hashende Datenblock erhält eine Kennung, welche dem Kommando mitteilt, dass mit diesem Datenblock der Hash-Vorgang nun abgeschlossen ist.

Dieses Kommando lässt aber noch mehr Varianten zu. Der berechnete Hash-Wert kann entweder in der Antwort auf das Kommando unmittelbar an das Terminal zurückgegeben werden, oder er wird zur Verwendung durch ein nachfolgendes Kommando in der Chipkarte gespeichert. Das Padding und der zu verwendende Schlüssel werden analog den vorhergehenden Kommandos mit `MANAGE SECURITY ENVIRONMENT` bei Bedarf eingestellt.

Tabelle 7.37 Die Funktionalität von `PERFORM SECURITY OPERATION` mit der Option `HASH` nach ISO/IEC 7816-8

PERFORM SECURITY OPERATION	
Option: HASH	
Kommando	• zu hashende Daten • Hash-Wert (falls Hash-Berechnung nur zum Teil in der Chipkarte) • <i>Schalter</i>: Hash-Berechnung nur zum Teil in der Chipkarte • <i>Schalter</i>: berechneten Hash-Wert in der Antwort zurückgeben • <i>Schalter</i>: letztes Kommando mit zu hashenden Daten
Antwort	• Hash-Wert (falls via Schalter gewählt) • Returncode

Die Option `COMPUTE DIGITAL SIGNATURE` kann zum Signieren von Daten benutzt werden. Dabei muss der Chipkarte der zu signierende Datenstring, üblicherweise zu einem Hash-Wert komprimiert, übergeben werden, oder er befindet sich bereits durch ein vorab ausgeführtes `PERFORM SECURITY OPERATION` mit der Option `HASH` in der Chipkarte. Diese Kommandooption lässt ebenfalls zu, dass unmittelbar die zu signierenden Daten übergeben werden. Diese können dann vor der eigentlichen Signatur noch in der Chipkarte gehasht werden. Für große Datenmengen ist ebenfalls, übereinstimmend mit `HASH`, ein Schicht-7-Chaining vorgesehen.

Falls die Datenlänge des Hashwertes nicht der Eingangslänge des Public-Key-Algorithmus entspricht, muss ein Padding stattfinden, welches in seinen Optionen wiederum durch `MANAGE SECURITY ENVIRONMENT` parametrisiert werden kann. Analog verhält es sich mit den zu verwendenden Schlüsseln.

Tabelle 7.38 Die Funktionalität von `PERFORM SECURITY OPERATION` mit der Option `COMPUTE DIGITAL SIGNATURE` nach ISO/IEC 7816-8

PERFORM SECURITY OPERATION	
Option: COMPUTE DIGITAL SIGNATURE	
Kommando	• zu signierende Daten <i>oder</i> Hash-Wert der zu signierenden Daten • <i>Schalter</i>: letztes Kommando mit zu signierenden Daten
Antwort	• digitale Signatur • Returncode

Die Prüffunktion für COMPUTE DIGITAL SIGNATURE ist die Kommandooption VERIFY DIGITAL SIGNATURE. Zur Überprüfung der digitalen Signatur könnte im Prinzip jeder hinreichend schnelle Computer verwendet werden, da der dazu benötigte Schlüssel öffentlich ist. Doch in vielen Fällen muss vorab noch die Gültigkeit des öffentlichen Schlüssels anhand einer weiteren digitalen Signatur überprüft werden, was wiederum durchaus sicherheitsrelevant ist und nicht auf einem ungeschützten Rechner durchgeführt werden sollte. Zur Prüfung der digitalen Signatur muss der Chipkarte entweder implizit der dazugehörige öffentliche Schlüssel bekannt sein oder explizit mit der nachfolgend beschriebenen Kommandooption VERIFY CERTIFICATE bekannt gemacht worden sein. VERIFY DIGITAL SIGNATURE können entweder direkt die zu prüfenden Daten oder der dazugehörige Hash-Wert übergeben werden. Alle anderen Parameter sind identisch mit COMPUTE DIGITAL SIGNATURE.

Tabelle 7.39 Die Funktionalität von PERFORM SECURITY OPERATION mit der Option VERIFY DIGITAL SIGNATURE nach ISO/IEC 7816-8

PERFORM SECURITY OPERATION	
Option: VERIFY DIGITAL SIGNATURE	
Kommando	• zu prüfende Daten <i>oder</i> Hash-Wert der zu prüfenden Daten • digitale Signatur • <i>Schalter</i>: letztes Kommando mit zu prüfenden Daten
Antwort	• Returncode

In offenen Systemen sind die öffentlichen Schlüssel für die Prüfung digitaler Signaturen üblicherweise mit dem geheimen Schlüssel der Zertifizierungsinstanz unterschrieben. Vor der Benutzung eines öffentlichen Schlüssels muss also dessen Echtheit geprüft werden, denn nur dann ist sichergestellt, dass dieser nicht gefälscht ist. Die Prüfung muss in einer sicheren Umgebung, z. B. in einer Chipkarte, stattfinden, da sie sonst manipuliert werden könnte.

Die Kommandooption VERIFY CERTIFICATE ist speziell für diese Prüfung von signierten öffentlichen Schlüsseln, d. h. Zertifikaten, vorgesehen. Wurde der öffentliche Schlüssel als echt identifiziert, kann er entweder dauerhaft in der Chipkarte gespeichert oder in dem direkt nachfolgenden VERIFY DIGITAL SIGNATURE als Schlüssel benutzt werden.

Tabelle 7.40 Die Funktionalität von PERFORM SECURITY OPERATION mit der Option VERIFY CERTIFICATE nach ISO/IEC 7816-8

PERFORM SECURITY OPERATION	
Option: VERIFY CERTIFICATE	
Kommando	• Zertifikat • <i>Schalter</i>: öffentlichen Schlüssel speichern
Antwort	• Returncode

Der folgende Ablauf zeigt die Anwendung der oben dargestellten Kommandos bei der Erstellung einer digitalen Signatur und einer nachfolgenden Prüfung.

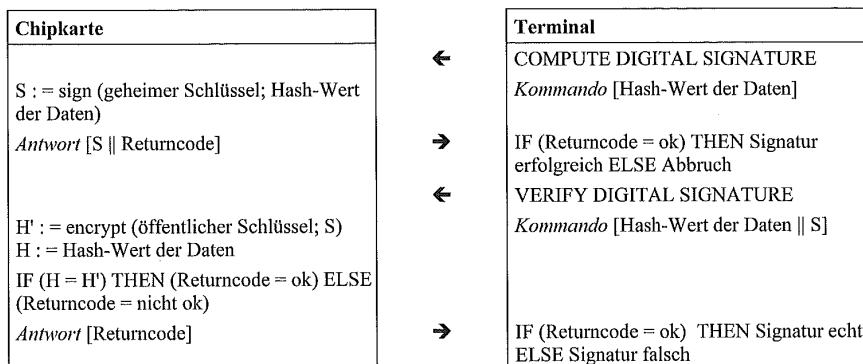


Bild 7.17 Beispiel für eine Kommandosequenz bei PERFORM SECURITY OPERATION mit den Kommandooptionen COMPUTE DIGITAL SIGNATURE und VERIFY DIGITAL SIGNATURE. Die Rahmenparameter (Schlüssel, Algorithmus, ...) sind entweder implizit vorgegeben oder vorab mit MANAGE SECURITY ENVIRONMENT eingestellt worden.

Unterstützt ein Chipkarten-Betriebssystem die Generierung von Schlüsselpaaren für asymmetrische Kryptoalgorithmen, so kann dieser Vorgang mit dem ISO/IEC 7816-8 Kommando GENERATE PUBLIC KEY PAIR angestoßen werden. Vorab sind alle für die Schlüsselerzeugung notwendigen Parameter mit MANAGE SECURITY ENVIRONMENT zu setzen.

Tabelle 7.41 Die Funktionalität von GENERATE PUBLIC KEY PAIR nach ISO/IEC 7816-8

GENERATE PUBLIC KEY PAIR	
Kommando	• ---
Antwort	• Returncode

7.8 Kommandos zur Verwaltung von Dateien

Die meisten neueren Chipkarten-Betriebssysteme erlauben es, innerhalb der Grenzen festgelegter Sicherheitsbedingungen Dateien zu vergrößern, neu anzulegen, zu löschen, zu sperren oder andere Verwaltungsfunktionen darauf auszuführen.

Allerdings wird manchmal bei Chipkarten mit nur einer Anwendung oft auf die meisten oder sogar alle Verwaltungsfunktionen verzichtet, da sie im Allgemeinen sehr Programmcode-intensiv sind und somit den Speicherbedarf und in der Folge die Kosten erhöhen. Bei Multiapplication-Karten ist es aber zwingend erforderlich, gewisse Verwaltungsfunktionen zu unterstützen, damit nicht alle Anwendungen gleichzeitig bei der Personalisierung in die Chipkarte geladen werden müssen.

Die Kommandos zur Verwaltung dürfen vom Standpunkt der Systemsicherheit aus nur nach vorheriger gegenseitiger Authentisierung ausgeführt werden, da sie ein idealer Ansatzpunkt für einen Angriff sind. Man denke sich dabei nur den Fall, dass ein Angreifer eine Datei zur Speicherung geheimer Informationen löscht und danach eine neue gleichen Namens erzeugt. Er setzt jedoch in der neuen Datei für das Lesen keine Zugriffsbedingung mehr, sodass sie ohne weitere Schutzmechanismen auslesbar ist. Für das Terminal existiert die Datei nach wie vor unter gleichem Namen, und es schreibt die geheimen Daten weiterhin in die manipulierte Datei. Dieser Angriff ist keineswegs neu, sondern existiert in etwas abgewandelter Form schon seit vielen Jahren. Er führt jedoch im Bereich von Dateiverwaltungen immer wieder zum Erfolg.

Einen anderen Angriffspunkt stellt die Benutzung von Verwaltungsfunktionen an öffentlich zugänglichen, also prinzipiell unsicheren Terminals dar. Hier muss in jedem Fall die Datenübertragung mit Secure-Messaging-Funktionen geschützt sein, denn nur dann eröffnet sich für einen Anwendungsanbieter die logistisch gesehen hervorragende Möglichkeit, Dateien und Anwendungen z. B. über öffentliche Kartentelefone auf sichere Weise in Chipkarten nachzuladen.

Speziell bei Multiapplication-Chipkarten, die von mehreren Anwendungsanbietern benutzt werden können, ist es notwendig, vor dem Erzeugen der einzelnen Anwendungen eine Speicheraufteilung sowie eine Zuweisung der Berechtigungsschlüssel zum Anlegen von Dateien zu machen. Damit verhindert man, dass ein einzelner Anwendungsanbieter den gesamten zur Verfügung stehenden Speicherplatz für seine Anwendung allokiert und kein Platz mehr für andere Anwendungen vorhanden ist.

Um bei diesem Problem Abhilfe zu schaffen, existiert ein Verfahren, mit dem einer Anwendung vorab Speicherplatz zugewiesen werden kann und dabei zusätzlich noch ein karten- und anwendungsspezifischer Schlüssel zur Erzeugung von Dateien in die Karte eingebracht wird. Dazu benutzt man das nicht genormte Kommando REGISTER. Anschließend kann man bei Kenntnis des anwendungsspezifischen Schlüssels neue Dateien erzeugen. Das Ergebnis dieses Verfahrens ist eine strikte Trennung der Vergabe von Speicherplatz und der Einbringung von neuen Dateien in die Chipkarte. Somit kann ein Kartenherausgeber einer Multiapplication-Chipkarte an mehrere Anwendungsanbieter Speicherplatz verkaufen, ohne befürchten zu müssen, dass diese ihn sich gegenseitig wegnehmen.

Tabelle 7.42 Die Funktionalität des nicht genormten Kommandos REGISTER

REGISTER	
Kommando	• DF Name des neuen DF • maximaler Speicherplatz für die neue Anwendung (d. h. das DF) • Schlüssel zum Anlegen der Dateien (d. h. für CREATE FILE)
Antwort	• Returncode

Das Kommando CREATE FILE ermöglicht das Erzeugen eines DFs oder EFs nach der Kompletierung der Chipkarte. Dabei ist es notwendig, dass ein bestimmter logischer Zustand, z. B. nach erfolgreicher gegenseitiger Authentisierung, erreicht sein muss. Je nach Umgebung, in der ein CREATE FILE ausgeführt wird, muss die Datenübertragung mit Secure Messaging geschützt sein. Die internationale Norm ISO/IEC 7816-9 definiert das CREATE FILE Kommando.

Tabelle 7.43 Die Funktionalität von CREATE FILE nach ISO/IEC 7816-9

CREATE FILE	
Kommando	• Typ der neuen Datei • IF (Dateityp = DF) THEN [DF Name der neuen Datei] IF (Dateityp = EF) THEN [• FID der neuen Datei • Zugriffsbedingungen • Struktur der neuen Datei] IF (Dateistruktur = transparent) THEN [• Größe der Datei] IF (Dateistruktur = linear fixed) OR (Dateistruktur = cyclic) THEN [• Anzahl der Records • Länge eines Records] IF (Dateistruktur = linear variable) THEN [• Anzahl der Records • Länge jedes einzelnen Records]
Antwort	• Returncode

Nachdem die Datei mit allen ihren Zugriffsbedingungen, Attributen und sonstigen Eigenschaften erzeugt wurde, kann sie mit SELECT FILE ausgewählt und dann darauf zugegriffen werden. Dabei muss das Betriebssystem verhindern, dass durch einen Abbruch des Erzeugungsvorgangs eine halb erzeugte Datei zu einem Angriff benutzt werden kann. Zusätzlich darf es nicht möglich sein, dass alte Speicherinhalte, die durch die neu erzeugten Dateien nur teilweise überschrieben wurden, ausgelesen werden können.

Der Header einer Datei enthält die kompletten Zugriffsbedingungen auf diese Datei. Dort sind beispielsweise Informationen darüber gespeichert, in welchem Zustand ein READ oder ein UPDATE auf den Dateiinhalt erfolgen darf. Gerade zum Zeitpunkt der Personalisierung der Chipkarte oder bei ausgedehnten Verwaltungsprozeduren im Dateibaum ist es von großem Vorteil, auf Dateien direkt ohne den Objektschutz zugreifen zu können. Dazu gab es in einem Entwurf der ISO/IEC 7816-9 ein Kommando MANAGE ATTRIBUTES um die Zugriffsbedingungen an einer vorab selektierten Datei zu verändern. Dieses Kommando durfte nur nach einer gegenseitigen Authentisierung und in einer sicheren Umgebung ablaufen. Durch die Einführung von Security Environments und die regelbasierten Zugriffsbedingungen nach ISO/IEC 7816-9⁴ wurde dieses Kommando jedoch überflüssig und ist nicht mehr in der Norm enthalten. Dennoch wird es aus Gründen der Einfachheit der Realisierung von einigen Chipkarten-Betriebssystemen unterstützt.

⁴ Siehe auch Abschnitt 5.9 „Ressourcenzugriffe nach ISO/IEC 7816-9“.

Tabelle 7.44 Die Funktionalität des mittlerweile obsoleten Kommandos MANAGE ATTRIBUTES nach einem Entwurf von ISO/IEC 7816-9

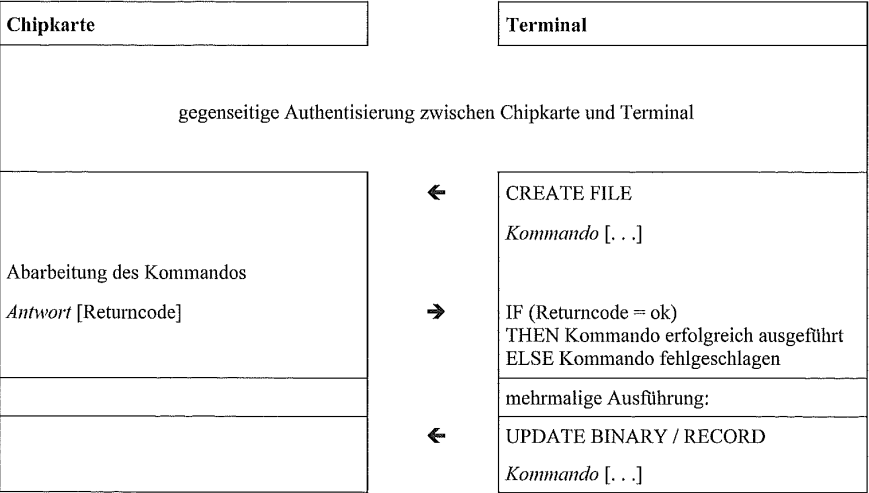
MANAGE ATTRIBUTES	
Kommando	• Neue Zugriffsbedingungen
Antwort	• Returncode

Falls das Kommando REGISTER vorhanden ist, benutzt man ihn beim Kartenherausgeber, um den maximal zu allozierenden Speicherplatz für die spätere Anwendung festzulegen. Außerdem wird schon ein temporärer DF Name, welcher einen AID (*application identifier*) enthalten kann, für das DF der Anwendung festgelegt und ein Schlüssel für CREATE FILE eingebracht. Mit dessen Kenntnis ist die nachfolgende Erzeugung von Dateien möglich.

Nun erhält der Benutzer die so vorbereitete Chipkarte. Falls er es für notwendig erachtet, kann er bei einem Anwendungsanbieter zusätzlich noch eine weitere Anwendung, z. B. für Kartentelefon, auf seiner Chipkarte nachladen lassen. Der Anwendungsanbieter benötigt dafür aber den geheimen Nachladeschlüssel, den er natürlich nur dann vom Kartenherausgeber bekommt, wenn ein vertragliches Verhältnis zwischen beiden besteht.

Nach einer erfolgreichen gegenseitigen Authentisierung zwischen Chipkarte und Terminal darf dann der Anwendungsanbieter mit CREATE FILE seine Dateien in dem ihm zugeteilten DF erzeugen. Dieser Vorgang könnte sowohl beim Anwendungsanbieter oder auch z. B. über ein öffentliches Kartentelefon erfolgen. Anschließend füllt er die EFs mit den notwendigen Daten und Schlüsseln und setzt die Zugriffsattribute für die Dateien. Danach ist die Anwendung betriebsbereit, und der Benutzer kann die neu gewonnene Funktionalität nutzen.

Der Ablauf zum Anlegen einer neuen Anwendung im Dateibaum einer Chipkarte gestaltet sich wie folgt:



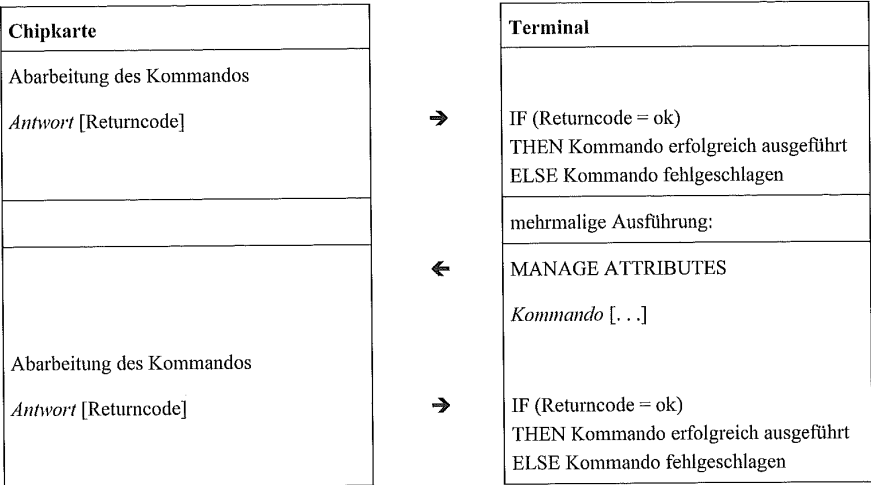


Bild 7.18 Beispiel für einen möglichen Ablauf beim Anlegen einer Anwendung.

Das Kommando DEACTIVATE FILE nach ISO/IEC 7816-9 bzw. INVALIDATE nach GSM 11.11 ermöglicht dem Terminal, eine vorher selektierte Datei reversibel zu sperren. Lese- und Schreibzugriffe auf die Datei sind danach nicht mehr möglich. Lediglich eine Selektion ist noch erlaubt.

Die EMV sieht ein ähnliches Kommando zum umkehrbaren Sperren von Anwendungen vor – APPLICATION BLOCK. Es sperrt nicht die EFs innerhalb der Anwendung, sondern nur die Kommandos für Selektion, Authentisierung und Zahlungsverkehr innerhalb des Anwendungs-DFs. Ansonsten verhält sich APPLICATION BLOCK ähnlich wie DEACTIVATE FILE und INVALIDATE.

Tabelle 7.45 Die Funktionalität von DEACTIVATE FILE nach ISO/IEC 7816-9

DEACTIVATE FILE		
Kommando	•	---
		oder
		FID
		oder
		Short-FID
		oder
		DF Name
Antwort	•	Returncode

Tabelle 7.46 Die Funktionalität von INVALIDATE nach EN 726-3

INVALIDATE		
Kommando	•	FID
Antwort	•	Returncode

Die Umkehrfunktionen zu DEACTIVATE FILE und INVALIDATE sind die Kommandos REACTIVATE FILE und REHABILITATE, mit denen eine gesperrte Datei wieder entsperrt werden kann. Dazu muss sie aber vor der Ausführung des Kommandos selektiert sein. Dass alle diese Kommandos nur in einem bestimmten Sicherheitszustand erlaubt sein dürfen, ist trivial, da man andernfalls jede Datei nach Belieben sperren oder wieder entsperren könnte. Um den Zugang zu einer Anwendung wieder zu öffnen, ist in der EMV das Kommando APPLICATION UNBLOCK festgelegt, das eine mit APPLICATION BLOCK gesperrte Anwendung wieder freigibt.

Tabelle 7.47 Die Funktionalität von REACTIVATE FILE nach ISO/IEC 7816-9

REACTIVATE FILE	
Kommando	• --- oder FID oder Short-FID oder DF Name
Antwort	• Returncode

Tabelle 7.48 Die Funktionalität von REHABILITATE nach GSM 11.11

REHABILITATE	
Kommando	• ---
Antwort	• Returncode

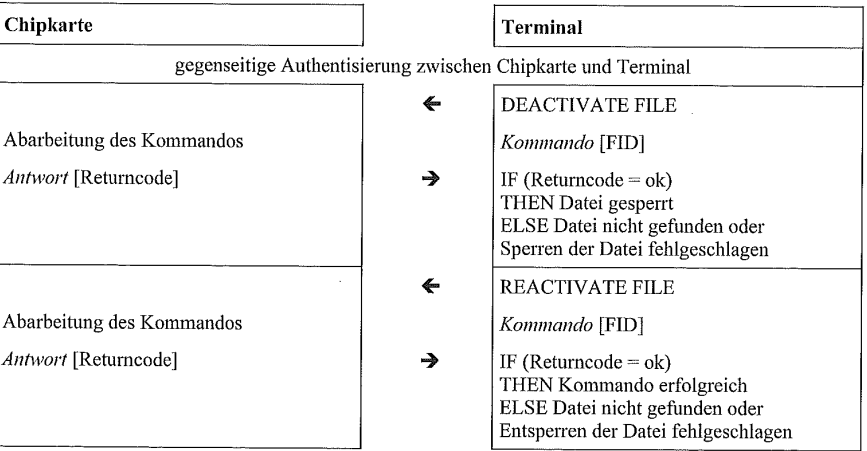


Bild 7.19 Beispiel für einen Kommandoablauf bei DEACTIVATE FILE und REACTIVATE FILE.

Das Kommando LOCK ist die nicht reversible Variante von INVALIDATE. Eine mit LOCK gesperrte Datei kann also nicht mehr entsperrt werden. Der Zustand ist vollständig irreversibel. Eine Variante der Benutzung ist die endgültige Sperrung einer Anwendung, deren Ablaufdatum erreicht wurde. Nach Ausführung eines LOCK-Kommandos kann die Datei nur mehr selektiert werden, jedes andere Zugriffskommando wird vom Betriebssystem unterbunden.

Tabelle 7.49 Die Funktionalität von LOCK nach EN 726-3

LOCK	
Kommando	• ---
Antwort	• Returncode

Das endgültige, also irreversible Sperren von Dateien hat den großen Nachteil, dass wertvoller Speicher in der Karte ein für alle Mal blockiert ist. Wesentlich eleganter ist es, bei nicht mehr benötigten Dateien den belegten Speicherplatz zu löschen und anderen oder neuen Anwendungen zugänglich zu machen.

Wichtig dabei ist, dass neben dem Entfernen der Datei aus dem Dateibaum der gesamte von der Datei benutzte Speicher physikalisch gelöscht wird. Nur dadurch kann man sicherstellen, dass alle Inhalte der Datei, die durchaus noch schützenswert und geheim sein können, überschrieben und niemandem mehr zugänglich sind. Wenn der beim Löschen frei werdende Speicher neuen Dateien zugute kommen soll, ist das Löschen einer Datei kompliziert und aufwendig. Deshalb ist dies nicht in allen Betriebssystemen vollständig verwirklicht.

Das Kommando DELETE FILE ist prinzipiell genauso zu handhaben wie die obig beschriebenen Kommandos zum Sperren und Entsperren von Dateien. Eine implizit mit dem Kommando selektierte Datei kann vollständig aus dem Dateispeicher der Chipkarte entfernt werden. Ob nun dieser frei werdende Speicher von anderen Dateien benutzt werden kann oder nicht, hängt vom jeweiligen Betriebssystem ab. Im Regelfall steht keine Freispeicherverwaltung zur Verfügung, und der Speicherplatz bleibt nach Ausführung des Kommandos ein für alle Mal verloren.

Tabelle 7.50 Die Funktionalität von DELETE FILE nach ISO/IEC 7816-9

DELETE FILE	
Kommando	• FID oder DF Name
Antwort	• Returncode

Zur unwiederbringlichen Sperrung eines DFs sieht die ISO/IEC 7816-9 das Kommando TERMINATE DF vor. Dabei bleibt das DF selektierbar, die in ihm enthaltenen Funktionen (z. B.: Programmcode) und Dateien sind jedoch nicht mehr zugänglich. Dieses Kommando kann zum „Abschalten“ einer Anwendung benutzt werden, wobei jedoch das frühere Vorhandensein eben dieser Anwendung nachvollziehbar bleibt.

Tabelle 7.51 Die Funktionalität von TERMINATE DF nach ISO/IEC 7816-9

TERMINATE DF	
Kommando	FID oder DF Name
Antwort	Returncode

Vergleichbar dem ISO/IEC 7816-9 Kommando TERMINATE DF verhält sich TERMINATE CARD USAGE, nur dass dieses Kommando die gesamte Chipkarte sperrt und danach alle weiteren Kommandos abgeblockt werden. Lediglich im ATR wird fortan der neue Kartenzustand angezeigt. Das EMV Äquivalent dazu ist das Kommando CARD BLOCK, welches ansonsten eine ähnliche Funktionalität aufweist.

Tabelle 7.52 Die Funktionalität von TERMINATE CARD USAGE nach ISO/IEC 7816-9

TERMINATE CARD USAGE	
Kommando	---
Antwort	Returncode

7.9 Kommandos zur Verwaltung von Applets

Bei Chipkarten-Betriebssystemen mit nachladbarem Programmcode ist es notwendig, diesen zum einen sicher und unter den Rahmenbedingungen des Kartenherausgebers in den Speicher der Chipkarte zu laden. Dazu existieren eine Reihe unterschiedlichster, doch meist firmenspezifischer Konzepte. Der einzige weltweit eingeführte Standard dazu ist die Open Platform (OP) Spezifikation von Global Platform [GP].

Mit dem Kommando LOAD wird eine Anwendung durch Übertragung der Ladedatei in die Chipkarte geladen. Das darauf aufbauende Kommando INSTALL etabliert die geladene Anwendung durch Aufrufen verschiedener Oncard-Funktionen des Card Managers und der Security Domain. Das Kommando DELETE wird zum Löschen von eindeutig identifizierbaren Objekten auf einer Chipkarte wie Ladedatei, Anwendung oder Schlüssel verwendet. Eine Beschreibung zu OP findet sich im Abschnitt 5.11 Open Platform.

7.10 Kommandos zur Komplettierung des Betriebssystems

Bei der Produktion der Chipkarten-Mikrocontroller wird lediglich das ROM programmiert, der EEPROM-Speicher bleibt bis auf eine Chipnummer und einen kartenindividuellen Schlüssel leer. Nachdem die Chipkarte aus Kartenkörper und Modul zusammengebaut wurde, muss das Betriebssystem im ROM durch die EEPROM-Teile ergänzt, d. h. komplettiert werden. Erst dann befindet sich ein vollständiges Betriebssystem mit seiner ganzen Funktionalität in der Chipkarte.

Um diese Teile in das EEPROM zu schreiben, existiert im ROM-Code ein relativ einfaches Ladeprogramm, mit dem nach vorheriger Schlüsselüberprüfung Daten in das

EEPROM geschrieben werden können. Die Adressierung des EEPROM-Speichers findet dabei linear byte- oder seitenweise mit direkten physikalischen Adressen statt.

Nachdem alle notwendigen Daten auf diese Weise in das EEPROM eingebracht wurden, wird das Betriebssystem der Karte vom reinen ROM-Betrieb umgeschaltet. Nunmehr laufen auch Prozesse und Routinen im EEPROM ab. Die Umschaltung kann mit einem Kommando durchgeführt werden, dessen Ausführungsbedingung nach vorherigem Vergleich einer Prüfsumme über die ergänzten EEPROM-Daten erfüllt ist. Durch die Prüfsumme stellt man sicher, dass auch alle Daten korrekt im EEPROM stehen. Für die Komplettierung verwendet man keine allzu komplexen Funktionen oder Authentisierungsverfahren, da man sich auf den ROM-Teil des Betriebssystems verlassen muss und es dort selbst kleinste Fehler unmöglich machen könnten, die Chipkarte zu komplettieren. Dies wäre ein zeitaufwendiger und in der Summe sehr teurer Fehler.

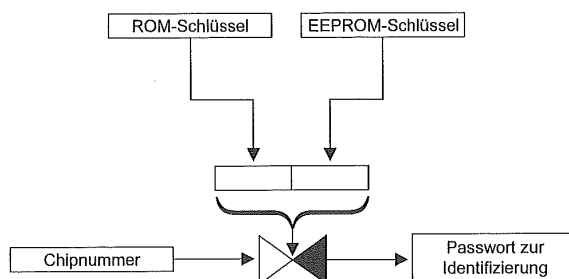


Bild 7.20 Eine von vielen Möglichkeiten zur Erzeugung eines chipindividuellen Passwortes zur Autorisierung der Komplettierung des Betriebssystems.

In den folgenden Teilen sind beispielhaft die drei notwendigen Kommandos für die Komplettierung eines Chipkarten-Betriebssystems aufgeführt. Sie weichen je nach Betriebssystem und Chipkartenhersteller stark voneinander ab. Es wird hier lediglich die notwendige Funktionalität aufgeführt. Allerdings benutzen praktisch alle Chipkarten diese oder eine ähnliche Methode, um das Betriebssystem zu vervollständigen. Das Kommando COMPARE KEY überprüft ein an die Karte übergebenes Passwort mit dem in ROM und EEPROM vom Halbleiterhersteller bei der Produktion eingebrachten Referenzpasswort. Der Schlüssel ist kartenindividuell und hat eine große Länge im Bereich um 32 Byte. Bei positivem Ergebnis sind nachfolgende Ladekommandos erlaubt. Bei negativem Vergleich wird ein Fehlbedienungs-zähler inkrementiert, der ab einem definierten Wert (z. B. 3) die Chipkarte gegen jegliche Zugriffe sperrt. Die Chipkarte kann dann dem Recycling zugeführt werden, da außer dem ATR keine Funktionalität mehr vorhanden ist.

Tabelle 7.53 Die Funktionalität von COMPARE KEY

COMPARE KEY	
Kommando	• Passwort
Antwort	• Returncode

Nachdem mit COMPARE KEY das Ladepasswort erfolgreich überprüft wurde, können mit WRITE DATA alle notwendigen Daten ins EEPROM geschrieben werden. Es ist dann möglich, das gesamte EEPROM byteweise zu adressieren und zu schreiben. Das bedeutet, dass neben den Betriebssystemdaten auch komplette Anwendungen eingebracht werden können. Dies ist im Übrigen die reguläre Methode zum Laden von Anwendungen bei Chipkarten mit sehr kleinem Speicher, in dem keine aufwendigen CREATE FILE-Kommandos mit dem dazugehörigen Zustandsautomaten Platz haben.

Tabelle 7.54 Die Funktionalität von WRITE DATA

WRITE DATA	
Kommando	• Daten • Speicheradresse im EEPROM
Antwort	• Returncode

Ist der verfügbare ROM-Speicher in der Chipkarte so klein, dass auch keine Testkommandos für das EEPROM darin Platz finden, besteht die Möglichkeit, sie mit diesen Kommandos in ihrer Grundfunktionalität zu simulieren. Dazu wird eine bestimmte Speicheradresse so lange beschrieben, bis die Chipkarte einen Schreibfehler meldet. Wurde die Zahl der Schreibvorgänge mitgezählt, dann ist damit die Anzahl der möglichen Schreib-/Löschzyklen bekannt. Dies ist die Hauptaussage, die von einem EEPROM-Testkommando im Rahmen der Qualitätssicherung erwartet wird.

Tabelle 7.55 Die Funktionalität von COMPLETION END

COMPLETION END	
Kommando	• Prüfsumme über EEPROM
Antwort	• Returncode

Nachdem mit einem oder mehreren aufeinander folgenden WRITE DATA-Kommandos alle Daten ins EEPROM geschrieben wurden, prüft man den korrekten Inhalt des EEPROMs und schließt die Komplettierung ab. Dazu findet das Kommando COMPLETION END Verwendung. Nach erfolgreichem Abschluss des Kommandos wird üblicherweise ein Reset auf der Chipkarte ausgelöst, damit das Betriebssystem neu initialisiert wird und so der neue Zustand erreicht werden kann.

Für Tests und Versuche ist es oftmals notwendig, eine bestehende Komplettierung wieder zu löschen. Dies ist üblicherweise nicht möglich, es existiert jedoch vereinzelt ein Kommando, das diese Funktionalität beinhaltet. Mit DELETE COMPLETION kann die Komplettierung eines Chipkarten-Betriebssystems wieder rückgängig gemacht werden. Damit ist es nicht mehr notwendig, für jede neue Komplettierung auch eine neue Chipkarte zu benutzen. Dieses Kommando wird ausschließlich für Testzwecke verwendet.

Tabelle 7.56 Die Funktionalität von DELETE COMPLETION

DELETE COMPLETION	
Kommando	• ---
Antwort	• Returncode

Die korrekte Komplettierung eines Chipkarten-Betriebssystems kann mit dem Kommando CHECK COMPLETION durchgeführt werden. Dazu berechnet die Chipkarte eine Prüfsumme über die Daten der Komplettierung und vergleicht diese mit einem Referenzwert, der ebenfalls mit diesen Daten gespeichert wurde. Stimmen beide Prüfsummen überein, so ist die Komplettierung richtig.

Tabelle 7.57 Die Funktionalität von CHECK COMPLETION

CHECK COMPLETION	
Kommando	• ---
Antwort	• Returncode

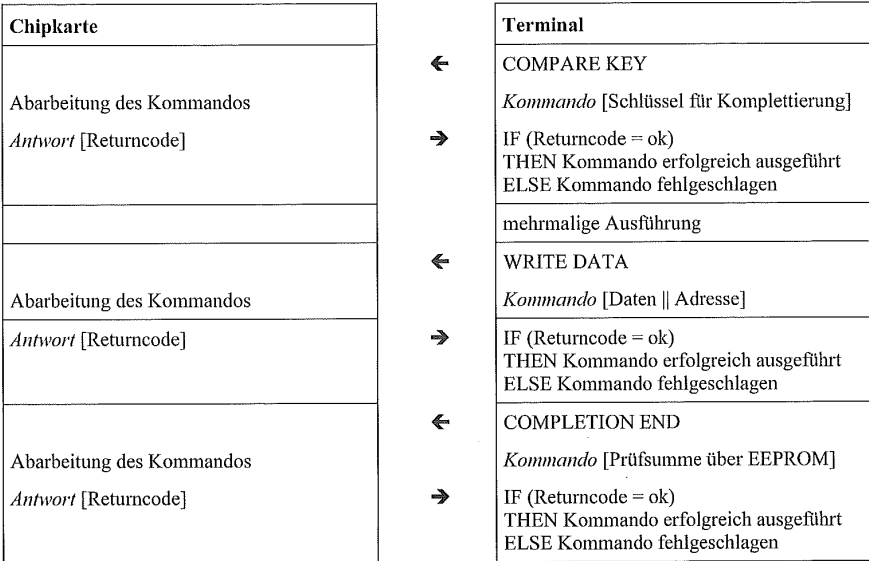


Bild 7.21 Beispiel für einen typischen Komplettierungsablauf.

Die vorgegebene Reihenfolge der Kommandos bei der Komplettierung verdeutlicht nochmals das folgende Zustandsdiagramm. Es wird von einem Automaten während der Komplettierung überwacht, sodass nur die aufgeführten Kommandos in der genau definierten Reihenfolge ausgeführt werden können.

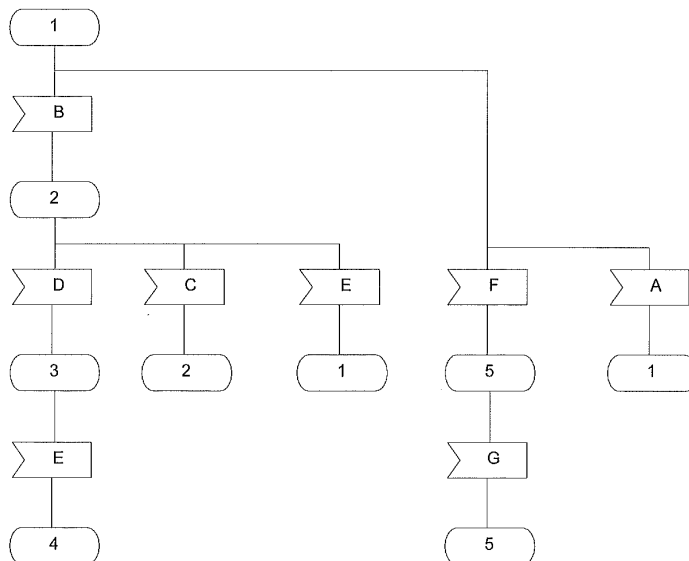


Bild 7.22 Zustandsautomat bei der Komplettierung eines Chipkarten-Betriebssystems.

- Zustände:
- 1 Grundzustand nach Reset der Chipkarte
 - 2 Chipkarte bereit zum Eintrag von Daten ins EEPROM
 - 3 Chipkarte komplettiert
 - 4 Grundzustand der Chipkarte nach der Komplettierung
 - 5 Chipkarte irreversibel gesperrt
- Übergänge:
- A alle Kommandos außer COMPARE KEY
 - B COMPARE KEY (erfolgreich ausgeführt)
 - C WRITE DATA
 - D COMPLETION END
 - E Reset der Chipkarte
 - F COMPARE KEY (3-mal nicht erfolgreich ausgeführt)
 - G alle Kommandos und Reset

7.11 Kommandos zum Test der Hardware

Das Betriebssystem einer Chipkarte überprüft beim Initialisieren zum Teil implizit und zum Teil explizit diverse Teile der Hardware. Die hier aufgeführten Kommandos gehen aber weit über die direkt im Betriebssystem integrierten Selbsttestroutinen hinaus. Im Zuge einer produktionsbegleitenden Qualitätssicherung ist es erforderlich, speziell die kritischen Teile eines Mikrocontrollers zu überprüfen. Diese Tests beziehen sich stark auf das EEPROM, da hier erfahrungsgemäß die meisten Probleme auftreten. Die Funktionalität des Prozessors ist implizit überprüft, nachdem das Terminal den ATR korrekt empfangen konnte.

Da keine Norm für die Testkommandos existiert, unterliegen sie in Funktionalität und Codierung dem jeweiligen Betriebssystem-Hersteller oder sind sogar abhängig vom Betriebssystem selbst.

Die Testkommandos können fest in das ROM eingebunden sein. Allerdings ist es durchaus auch üblich, Testkommandos bei Bedarf mittels der Komplettierungskommandos ins EEPROM nachzuladen und dort auszuführen. Dass dies im Falle eines nicht voll funktionsfähigen EEPROMs zu Problemen führt, ist einleuchtend. Der Vorteil liegt natürlich in dem nunmehr größeren zur Verfügung stehenden ROM. Im Sinne eines sicheren Betriebssystems muss der Einsatz der Testkommandos auf die Phase vor der Komplettierung beschränkt sein. Dies bedeutet, dass alle Testkommandos bei einer initialisierten oder sogar personalisierten Karte gesperrt sind. Eine Ausnahme können dabei der RAM-Test und die Prüfsumme über ROM oder EEPROM bilden, da diese Kommandos die Sicherheit nicht beeinträchtigen.

Nachfolgend sind einige Kommandos aufgeführt, die für extensive Tests der Hardware benutzt werden können. Ein fehlerhaftes RAM würde noch vor dem ATR zu einem Totalabsturz des Betriebssystems führen, doch können auch nur einzelne Bits oder Bytes im RAM kaputt sein. Dies hätte dann nur Auswirkungen auf spezielle Funktionen oder Teile davon.

TEST RAM prüft das gesamte RAM und sendet eine entsprechende Antwort zum Terminal zurück. Beim Test müssen alle verfügbaren Bytes mit unterschiedlichen Testmustern beschrieben und gelesen werden. Typisch dabei ist ein abwechselndes Schreiben mit '55' und 'AA', da diese beiden hexadezimalen Werte auf der Bit-Ebene betrachtet ein Schachbrettmuster ergeben. Ideal ist auch ein Wellenformtest, bei dem der Speicher einmal von der niedrigen zur höheren Adresse wie auch umgekehrt beschrieben und wieder gelesen wird. Die genaue Implementierung dieses Tests ist vom jeweiligen Betriebssystem abhängig. Zum Teil wird auch ganz auf diesen Test verzichtet.

Tabelle 7.58 Die Funktionalität von TEST RAM

TEST RAM	
Kommando	• ---
Antwort	• Returncode

CALCULATE EDC ist ein sehr einfacher Test, der eine Prüfsumme (EDC) über das gesamte ROM bzw. zu definierende EEPROM-Teile errechnet und an das Terminal zurückgibt. Dies ist eine Methode, um festzustellen, ob die ROM-Maske unverändert ist oder ob Speicherzellen im EEPROM gekippt sind. Die Überprüfung des EEPROMs bezieht sich nur auf statische Bereiche, die innerhalb der Kartenlebensdauer nicht gewollt verändert werden können. Das Terminal vergleicht die von der Chipkarte erhaltene Prüfsumme mit einer vorhandenen Referenz und entscheidet, ob der Speicherinhalt noch konsistent ist.

Tabelle 7.59 Die Funktionalität von CALCULATE EDC

CALCULATE EDC	
Kommando	• <i>Schalter</i> : ROM / EEPROM
Antwort	• Prüfsumme
	• Returncode

Der EEPROM-Dauertest TEST EEPROM ist ein Test für eine die Speicherinhalte überschreibende Prüfung des EEPROMs. Die Chipkarte erhält zwei Muster, die sie alternierend in einen Speicherbereich schreibt. Die Bereichsgrenzen sowie die Anzahl der Schreibversuche können innerhalb gewisser Einschränkungen im Kommando festgelegt werden. Selbstverständlich muss das Betriebssystem nach jedem Schreibvorgang den Speicherinhalt auf Fehler überprüfen.

Tabelle 7.60 Die Funktionalität von TEST EEPROM

TEST EEPROM	
Kommando	• Muster 1 • Muster 2 • Anzahl der Schreibzyklen • Startadresse im EEPROM • Endadresse im EEPROM
Antwort	• Anzahl der ausgeführten Schreibzyklen (falls Fehler aufgetreten sind) • Fehleradresse (falls Fehler aufgetreten sind) • Returncode

Dadurch, dass der Chipkarte in einem Parameter die Anzahl der Schreibzyklen übergeben wurde, kann dieses Kommando intern abgearbeitet werden, was zu einem großen Geschwindigkeitsvorteil gegenüber einzelnen abgesandten Kommandos führt. Konnte die Chipkarte einen Schreibfehler entdecken, so sendet sie die Anzahl der schon ausgeführten Schreibzyklen sowie die Fehleradresse an das Terminal. Die Anwendung des EEPROM-Dauertests ist nicht nur auf die zerstörende Prüfung des EEPROMs beschränkt, da sich dieser Test auch zum wahlfreien Beschreiben von EEPROM-Bereichen verwenden lässt. Die Anzahl der Schreibzyklen wird dabei auf 1 gesetzt.

Tabelle 7.61 Die Funktionalität von COMPARE EEPROM

COMPARE EEPROM	
Kommando	• Vergleichswert • Adresse
Antwort	• Returncode

Mit dem EEPROM-Vergleichstest COMPARE EEPROM lässt sich dann überprüfen, ob das geschriebene Muster noch im EEPROM enthalten ist. Diese Kommando-Kombination findet vor allem für den Bereich der Tests des EEPROM-Datenerhalts bei verschiedenen Temperaturen Anwendung. Dabei beschreibt man mehrere Speicherseiten mit einem Muster, legt die Chipkarte in einen Klimaschrank und überprüft nach einer bestimmten Zeitspanne, ob der EEPROM-Inhalt noch vorhanden ist.

Mit TEST EEPROM ließe sich im Prinzip das gesamte EEPROM löschen, wenn man die Start- und Endadresse entsprechend setzt. Jedoch existiert in vielen Betriebssystemen

ein Kommando, das hier den Namen DELETE EEPROM hat. Mit ihm kann das gesamte EEPROM in einem Zug gelöscht werden, sodass alle Speicherinhalte überschrieben sind.

Die Anwendung beschränkt sich auf zwei Einsatzzwecke. Nachdem etliche Testkommandos ihre Prüfdaten im EEPROM hinterlassen haben, kann es mit wenig Aufwand wieder auf einen definierten Zustand gesetzt werden. Der zweite sinnvolle Einsatz erfolgt vor der Komplettierung des Betriebssystems bzw. vor der Initialisierung. Mit DELETE EEPROM können diese Prozesse beschleunigt werden, da ein bereits gelöscht EEPROM schneller geschrieben werden kann, weil das vor dem Schreiben oft notwendige Löschen von EEPROM-Pages entfällt.

Tabelle 7.62 Die Funktionalität von DELETE EEPROM

DELETE EEPROM	
Kommando	• ---
Antwort	• Returncode

7.12 Kommandos für Datenübertragung

Übertragungsprotokolle sollten prinzipiell völlig unabhängig von den Daten und Kommandos der Anwendungsschicht aufgebaut sein. Dies würde auch das Schichtenmodell nach OSI vorsehen. Leider klafft hier eine Lücke zwischen theoretischem Anspruch und der Praxis. Es gibt zwei Kommandos, deren einziger Zweck darin besteht, auf Anwendungsebene Mechanismen des Übertragungsprotokolls auszuführen. Es sind dies die Kommandos GET RESPONSE und ENVELOPE. Des Weiteren existiert noch das Kommando MANAGE CHANNEL, dessen Funktion ebenfalls nicht nur von der Anwendungsschicht benutzt wird.

Im T=0-Protokoll ist es nicht möglich, während eines Kommando-Antwort-Zyklus einen Datenteil sowohl zur Chipkarte zu senden als auch von der Chipkarte zu empfangen.⁵ Das Protokoll unterstützt damit also keine Case-4-Kommandos, die jedoch häufig benutzt werden. Deshalb muss man sich bei T=0 gezwungenermaßen mit einer Hilfskonstruktion behelfen. Die Funktionsweise ist einfach: Man sendet das Case-4-Kommando zur Chipkarte und erhält im Gutfall einen speziellen Returncode, der dem Terminal signalisiert, dass das Kommando Daten erzeugt hat, die noch abgeholt werden können. Daraufhin sendet das Terminal ein GET RESPONSE-Kommando zur Chipkarte und erhält nun die erzeugten Daten. Damit ist der Kommando-Antwort-Zyklus für das erste Kommando abgeschlossen. Solange kein anderes Kommando als GET RESPONSE zur Chipkarte gesendet wird, ist es erlaubt, die Antwortdaten mehrmals anzufordern.

Tabelle 7.63 Die Funktionalität von GET RESPONSE nach ISO/IEC 7816-4 und GSM 11.11

GET RESPONSE	
Kommando	• Anzahl der zu sendenden Daten
Antwort	• Daten
	• Returncode

⁵ Siehe auch Abschnitt 6.4.2 „Übertragungsprotokoll T = 0“.

Tabelle 7.64 Die Funktionalität von ENVELOPE nach ISO/IEC 7816-4

ENVELOPE	
Kommando	• Kommando-APDU
Antwort	• Antwort-APDU • Returncode

Werden im Rahmen von Secure Messaging Kommandos vollständig verschlüsselt, kann es bei dem Übertragungsprotokoll T = 0 zu Problemen bei der Übertragung kommen, da dieses sowohl ein unverschlüsseltes Instruction-Byte als auch ein unverschlüsseltes Le-Byte benötigt. Um diese Einschränkung zu umgehen, benutzt man ENVELOPE. Mit ihm kann eine vollständige APDU mit Header und Datenteil in den Datenteil der ENVELOPE APDU integriert werden. Dieser lässt sich sodann ohne Einschränkungen verschlüsseln und mit jedem Übertragungsprotokoll übertragen. Das Gleiche gilt für die von der Chipkarte erzeugte Antwort, die ebenfalls in die APDU von ENVELOPE eingebettet wird. Parallel dazu wird das ENVELOPE Kommando auch zur Realisierung von proaktiven Kommandos⁶ bei SAT und USAT benutzt. Diese Verwendung steht jedoch im Einklang zu obiger Beschreibung mit dem Unterschied, dass keine Verschlüsselung der Daten durchgeführt wird.

Mithilfe von logischen Kanälen können auf einer einzelnen Chipkarte bis zu vier Anwendungen unabhängig voneinander angesprochen werden.⁷ Die Zuordnung von Kommando und Anwendung findet dabei durch zwei Bits im Class-Byte statt. Vor Benutzung eines neuen logischen Kanals muss dies der Chipkarte explizit mitgeteilt werden. Dazu gebraucht man MANAGE CHANNEL. Mit ihm wird der Chipkarte signalisiert, dass ein weiterer logischer Kanal benötigt wird. Dazu kann entweder explizit die Nummer angegeben werden, oder das Terminal erhält von der Chipkarte nach einer prinzipiellen Anfrage die Nummer eines freien logischen Kanals.

Tabelle 7.65 Die Funktionalität von MANAGE CHANNEL nach ISO/IEC 7816-4

MANAGE CHANNEL	
Kommando	• Schalter: logischen Kanal öffnen/schließen • IF (bestimmter Kanal gewünscht) THEN • Nummer des logischen Kanals
Antwort	• Nummer des logischen Kanals (falls ein neuer logischer Kanal geöffnet wurde) • Returncode

Bei der Eröffnung eines neuen logischen Kanals vom Standardkanal mit der Nummer 0 aus, verhält sich die Chipkarte für diesen Kanal wie nach einem Reset, d. h. das MF ist selektiert, und es ist noch kein Sicherheitszustand erreicht. Wird von einem logischen Kanal, der nicht die Standardnummer 0 hat, ein neuer logischer Kanal eröffnet, so bleiben das aktuell selektierte DF sowie der Sicherheitszustand erhalten. Nach dem Schließen eines logischen Kanals sind die betreffende Dateiselektion und der Sicherheitszustand gelöscht.

⁶ Siehe auch Abschnitt 13.2.4 „Die SIM“.

⁷ Siehe auch Abschnitt 6.7 „Logische Kanäle“.

7.13 Datenbankkommandos – SCQL

Am Anfang waren Chipkarten Identifikationsmedien und als solche typische Einbenutzersysteme (*single user systems*). Die hoch entwickelten Sicherheits- und Zugriffsmechanismen von Multiapplication-Chipkarten könnten jedoch durchaus dazu verwendet werden, Chipkarten auch als Mehrbenutzersysteme (*multi user system*) zu konfigurieren und zu betreiben. Allerdings sollte der dafür notwendige Aufwand nicht unterschätzt werden. Auch erreichen die notwendigen kryptografischen Schutzmechanismen schnell eine bedenkliche Komplexität.

Man sollte dies analog zu den unterschiedlichen Dateistrukturen sehen. Prinzipiell kann mit einer Datei der Struktur „transparent“ auch ein datensatzorientiertes Telefonverzeichnis aufgebaut werden, doch mit einer „linear-variable“-Struktur funktioniert dies nun einmal wesentlich einfacher.

Dies war schlussendlich die Motivation, warum auf Chipkarten eine Teilmenge der Datenbankabfragesprache SQL (*structured query language*) als ISO/IEC 7816-7 genormt und auch von verschiedenen Betriebssystem-Herstellern in ihre Produktpalette aufgenommen wurde. Die für Chipkarten festgelegte Untermenge von SQL, welche in ISO/IEC 9075 festgelegt ist, hat den Namen SCQL (*structured card query language*).

Das Haupteinsatzgebiet von Chipkarten mit SCQL-Funktionalität wird im Gesundheitsbereich liegen, da dort unterschiedliche Personen und Organisationen auf Daten mit differierenden Lese- und Schreibrechten zugreifen müssen. Allerdings gibt es zurzeit keine größeren Anwendungen, bei denen SCQL eingesetzt wird.

SCQL unterstützt, genauso wie SQL, tabellenorientierte Datenbanksysteme. Diese bestehen aus einem Tabellennamen, einer festen Anzahl benannter Spalten und einer variablen Anzahl von Zeilen. Auf diese Tabelle lässt sich nun eine logische Sicht (*view*) festlegen. Ist sie statisch, dann bedeutet dies, dass eine bestimmte Anzahl von Spalten für diese Sicht fixiert ist. Eine dynamische Sicht ist dagegen eine Auswahl von Zeilen, die eine bestimmte Bedingung (z. B.: Vorname = „Wolfgang“) erfüllen. Kombinationen von statischen und dynamischen Sichten sind gestattet. Eine Sicht hat ebenfalls einen eigenen Namen und kann als Basis zum Lesen und Schreiben von Daten benutzt werden.

SCQL in Chipkarten benötigt den zusätzlichen Dateityp „Database“ im Betriebssystem. Diese Datei, DBF (*database file*) genannt, ist ein Datenbankobjekt direkt unter dem MF oder unter einem DF. Es enthält die Datentabellen, dazugehörige Systemtabellen, und es kann ohne vorherige Selektion von den Datenbank-Kommandos aus angesprochen werden. Ein DBF ist ein logisches Konstrukt und kann abhängig vom jeweiligen Chipkarten-Betriebssystem auch auf mehrere EFs verteilt sein.

Zur Benutzer- und Rechteverwaltung müssen im DBF drei Systemtabellen eingerichtet werden. In der Object Description Table sind Informationen über die Tabellen und die dazugehörigen Sichten gespeichert. Diese wird durch die User Description Table ergänzt, welche die Benutzer der Datenbank festlegt. In der dritten Tabelle des DBF, der Privilege Description Table, sind die Rechte der einzelnen Benutzer auf Tabellen und Sichten festgelegt, sowie deren erlaubte Operationen darauf.

Auf eine SCQL-Tabelle bzw. eine SCQL-Sicht können die Operationen Lesen, Einfügen, Schreiben und Löschen ausgeführt werden, welche über Zugriffsrechte reglementiert sind. Für Lese- und Schreiboperationen auf die Tabelle oder Sicht ist ein Cursor definiert. Er zeigt auf die entsprechende Zeile, welche dann für eine Aktion verwendet werden soll.

Tabelle		
Spalte 1	Spalte 2	Spalte 3
Zeile 1	Zeile 1	Zeile 1
Zeile 2	Zeile 2	Zeile 2
Zeile 3	Zeile 3	Zeile 3
Zeile 4	Zeile 4	Zeile 4
Zeile 5	Zeile 5	Zeile 5

Sicht 1 (statisch)	
Spalte 1	Spalte 2
Zeile 1	Zeile 1
Zeile 2	Zeile 2
Zeile 3	Zeile 3
Zeile 4	Zeile 4
Zeile 5	Zeile 5

Sicht 2 (dynamisch)		
Spalte 1	Spalte 2	Spalte 3
Zeile 3	Zeile 3	Zeile 3
Zeile 5	Zeile 5	Zeile 5

Bild 7.23 Eine SCQL-Tabelle mit drei Spalten und fünf Zeilen, auf die eine statische Sicht und eine dynamische Sicht eingerichtet wurde. Sicht 1 ist statisch und zeigt vollständig Spalte 1 und 2. Sicht 2 ist dynamisch und zeigt für alle drei Spalten die Zeilen, für die eine bestimmte, hier nicht näher dargestellte Abfragebedingung erfüllt ist.

Tabelle: Bekannte von Louis Wu		
Name	Volk	Protektor
Teela Brown	Kugelvolk	Ja
Nessus	Puppetier	Nein
Bram	Vampir	Ja
Chmee	Kzinti	Nein
Akolyth	Kzinti	Nein
Prill	Maschinenvolk	Nein
Vala	Maschinenvolk	Nein

Sicht: Protektoren	
Name	Volk
Teela Brown	Kugelvolk
Bram	Vampir

Sicht: Kzinti	
Name	
Chmee	
Akolyth	

Sicht: Name	
Teela Brown	
Nessus	
Bram	
Chmee	
Akolyth	
Prill	
Vala	

Bild 7.24 Beispiel für eine SCQL-Tabelle mit den Spalten Name, Volk, der Eigenschaft Protektor und sieben Einträgen. Zusätzlich sind noch drei Sichten eingezeichnet. Für diese können bei SCQL unterschiedliche Zugriffsrechte festgelegt werden.

Tabelle 7.66 Das SCQL Kommando `PERFORM SCQL OPERATION` nach ISO/IEC 7816-7 und seine Funktionen

PERFORM SCQL OPERATION	
CREATE TABLE	Erstellt eine neue Tabelle mit ihren Spalten und Spaltennamen.
CREATE VIEW	Erzeugt eine neue Sicht (statisch und/oder dynamisch) auf eine Tabelle.
CREATE DICTIONARY	Erzeugt die Object Description Table, die User Description Table und die Privilege Description Table.
DROP TABLE	Löscht eine Tabelle.
DROP VIEW	Löscht eine Sicht.

Tabelle 7.66 (Fortsetzung)

PERFORM SCQL OPERATION	
GRANT	Vergibt Zugriffsrechte für einen Einzelbenutzer, eine Benutzergruppe oder alle Benutzer.
REVOKE	Entzieht Zugriffsrechte, die vorher durch GRANT vergeben wurden.
DECLARE CURSOR	Legt einen Cursor fest, der auf eine Zeile in einer Tabelle, Sicht oder Systemtabelle deutet.
OPEN	Aktiviert einen Cursor auf die erste Zeile.
NEXT	Setzt einen Cursor auf die nächste Zeile.
FETCH	Liest die Zeile, auf die ein Cursor zeigt.
FETCH NEXT	Liest die logisch nächste Zeile, auf die ein Cursor zeigt. Der Cursor wird auf die gelesene Zeile gesetzt.
INSERT	Fügt am Ende einer Tabelle eine Zeile ein, ohne den Cursor zu verändern.
UPDATE	Schreibt Daten in ein oder mehrere Felder einer Zeile in einer Tabelle. Die Zeile wird durch den Cursor festgelegt.
DELETE	Löscht diejenige Zeile einer Tabelle, auf die der Cursor zeigt. Der Cursor wird auf die nächste logische Zeile gesetzt.

Tabelle 7.67 Das SCQL-Kommando „PERFORM TRANSACTION OPERATION“ nach ISO/IEC 7816-7 und seine Funktionen

PERFORM TRANSACTION OPERATION	
BEGIN	Reserviert Platz für ein Speicherabbild, beispielsweise eine Zeile der Tabelle, für die Funktionen COMMIT und ROLLBACK.
COMMIT	Prüft alle Änderungen einer Tabelle, die seit dem letzten BEGIN gemacht wurden.
ROLLBACK	Restauriert die Tabelle wieder in den Zustand, der vor dem letzten BEGIN herrschte.

Tabelle 7.68 Das SCQL-Kommando „PERFORM USER OPERATION“ nach ISO/IEC 7816-7 und seine Funktionen

PERFORM USER OPERATION	
PRESENT USER	Identifizierung eines Benutzers anhand seiner Benutzer-ID.
CREATE USER	Erzeugung eines Eintrags für einen neuen Benutzer.
DELETE USER	Löschen des Eintrags für einen Benutzer.

Es gibt drei grundlegende SCQL-Kommandos: PERFORM SCQL OPERATION, PERFORM TRANSACTION OPERATION und PERFORM USER OPERATION. Dazu werden lediglich drei Codes für die Instruction INS benötigt, da die eigentlichen SCQL-Operationen über das Parameterbyte P2 getriggert werden. Alle Daten im Kommando Body und Antwort Body sind TLV-codierte Datenobjekte. Die drei Kommandos und ihre Funktionen sind in den folgenden Tabellen überblickshaft dargestellt.

Obwohl SCQL gegenüber SQL viele Einschränkungen hat (keine Sortierungen, keine verschachtelten Abfragen, keine Joins, ...), besitzt es doch im Mehrbenutzerbereich durchaus Verwendungspotenzial für die Zukunft. Die Speicher der aktuell verfügbaren Mikrocontroller sind allerdings für extensive SCQL-Datenbanken auf Chipkarten noch nicht groß genug, sodass es durchaus einige Zeit dauern kann, bis erste größere Anwendungen vorliegen.

7.14 Kommandos für elektronische Geldbörsen

Die europäische Norm für branchenübergreifende elektronische Geldbörsen EN 1546 definiert in Teil 3 insgesamt sechs Kommandos für die Börse und zwölf für das Sicherheitsmodul im Terminal, welches ebenfalls eine Chipkarte sein kann. In diesem Unterkapitel sind die vier wichtigsten Kommandos für die Chipkarte mit einer elektronischen Geldbörse in ihrem grundlegenden Aufbau beschrieben.⁸ Mit diesen Kommandos kann auf einer Chipkarte eine Anwendung betrieben werden, mit der es möglich ist, mit einer vorbezahlten Börse bargeldlos zu bezahlen und diese auch wieder aufzuladen. Die Kommandos für Fehlerbehebung, Währungsumrechnung, Parameteränderung und Storno, sowie alle Kommandos für das Sicherheitsmodul, sind nicht weiter beschrieben. Die Spezifikation CEPS (*Common European Purse System*) für elektronische Geldbörsen legt sehr ähnliche Kommandos wie die der EN 1546 fest.

Die folgenden Beschreibungen würden auch in das Unterkapitel „7.14 Anwendungsspezifische Kommandos“ passen, da die dargestellten Kommandos für genau diese eine Anwendung definiert sind. Diese Kommandos können nie für andere Einsatzzwecke als für elektronische Geldbörsen benutzt werden, da sie dahingehend optimiert worden sind. Ihnen wurde aber hier ein eigenes Unterkapitel gewidmet, da elektronische Geldbörsen in Zukunft neben der Telekommunikation eine der Hauptanwendungen für Chipkarten sind.

Alle Transaktionen für eine elektronische Geldbörse sind nach EN 1546 dreigeteilt. Im ersten Teil wird mit einem Initialisierungskommando (INITIALIZE IEP for Load/for Purchase) die Börse initialisiert. Mit dem zweiten Kommando wird die eigentliche Transaktion ausgeführt (d. h. Laden der Börse oder Bezahlen mit der Börse). Im optionalen dritten Teil wird die vorangehende Transaktion bestätigt. Alle Börsenkommandos greifen direkt auf Dateien in der Börsenanwendung der Chipkarte schreibend und lesend zu. Dies sind Dateien, die diverse Parameter, den Börsensaldo und Protokollaufzeichnungen enthalten. Mit den folgenden Kommandos werden nun die einzelnen Teilschritte bei einer Börsentransaktion ausgeführt. Für jedes Kommando ist der genaue interne Ablauf – sowohl in der Funktionalität als auch in der Reihenfolge der einzelnen Schritte – in der EN 1546-2 genau festgelegt, sodass alle Implementierungen zumindest die gleichen Grobabläufe haben.

Das Kommando INITIALIZE IEP kann für mehrere Zwecke verwendet werden. Mit einem Parameter wird zwischen der Initialisierung von Ladetransaktionen, Bezahltransaktionen und weiteren Transaktionsarten umgeschaltet.

⁸ Kommandosequenzen und generelle Systemstrukturen eines elektronischen Geldbörsensystems sind im Abschnitt 12.3.1 „CEN-Norm EN 1546“ im Detail beschrieben.

Tabelle 7.69 Die Funktionalität von INITIALIZE IEP for Load nach EN 1546-3

INITIALIZE IEP for Load	
Kommando	• --- • Ladebetrag (M_{LDA}) • Währungscode ($CURR_{LDA}$) • PPSAM-Bezeichner (PPSAM) • Zufallszahl (R) • benutzerbestimmte Daten (DD)
Antwort	• Bezeichner des Börsenanbieters (PP_{IEP}) • IEP-Bezeichner (IEP) • verwendeter kryptografischer Algorithmus (ALG_{IEP}) • Verfallsdatum ($DEXP_{IEP}$) • Börsensaldo (BAL_{IEP}) • Transaktionsnummer der IEP (NT_{IEP}) • Schlüsselinformation (IK_{IEP}) • Signatur S_1 • Returncode (CC_{IEP})

Tabelle 7.70 Die Funktionalität von CREDIT nach EN 1546-3

CREDIT IEP	
Kommando	• Schlüsselinformation (IK_{PPSAM}) • Signatur S_2 • benutzerbestimmte Daten (DD)
Antwort	• Signatur S_3 • Returncode (CC_{IEP})

Das Kommando INITIALIZE IEP for Load leitet als ersten Schritt das Aufladen der Börse in der Chipkarte ein. Dazu werden die übergebenen Daten, wie beispielsweise Währungscode und Ladebetrag, in der Chipkarte geprüft, ob sie mit den vorgegebenen Werten in den Parameterdateien übereinstimmen. Zusätzlich lassen sich auch noch frei definierbare Daten (benutzerbestimmte Daten) in einer Protokolldatei speichern. Anschließend wird ein Transaktionszähler erhöht und eine Signatur S_1 über verschiedene Daten (z. B. aktueller Börsensaldo, Verfallsdatum) gebildet, sodass diese manipulationssicher zum Terminal übertragen werden können.

Im zweiten Schritt der Aufladetransaktion erhält die Chipkarte im Wesentlichen Informationen über zu verwendende Schlüssel und eine Signatur S_2 . Diese stammt vom Sicherheitsmodul des Terminals und ermöglicht der Chipkarte, neben dem Manipulationsschutz von Daten das Sicherheitsmodul zu authentisieren. Die Chipkarte ist durch das Sicher-

heitsmodul im Terminal durch ein vorangehendes INITIALIZE IEP for Load schon authentisiert. Nach erfolgreicher Prüfung von S_2 erhöht die Chipkarte den Saldo der Börse, ergänzt den aktuellen Record in der Protokolldatei und erzeugt zur Bestätigung eine Signatur S_3 . Dies wird dann vom Sicherheitsmodul im Terminal als Bestätigung für das korrekte Aufbuchen des Ladebetrags verwendet.

Der Zweite hier vorgestellte Ablauf mit dem dazu notwendigen Kommando zeigt das Bezahlen mit einer elektronischen Geldbörse. Als erstes Kommando zur Initialisierung der Transaktion wird hier ebenfalls ein INITIALIZE IEP verwendet, nur mit der Option „for Purchase“. Die Chipkarte erhält von diesem Kommando keine Daten, doch erhöht es in der Karte den Transaktionszähler. Anschließend wird eine Signatur S_1 über Daten wie Verfallsdatum, Transaktionszähler und IEP-Bezeichner gebildet. Diese so mit der Signatur für den Übertragungsweg gesicherten Daten werden mit einigen anderen zurück zum Terminal geschickt.

Tabelle 7.71 Die Funktionalität von INITIALIZE IEP for Purchase nach EN 1546-3

INITIALIZE IEP for Purchase	
Kommando	• ---
Antwort	• Bezeichner des Börsenanbieters (PP_{IEP}) • IEP-Bezeichner (IEP) • verwendeter kryptografischer Algorithmus (ALG_{IEP}) • Ablaufdatum ($DEXP_{IEP}$) • Börsensaldo (BAL_{IEP}) • Währungscode ($CURR_{IEP}$) • Authentisierungsmodus (AM_{IEP}) • Transaktionsnummer der IEP (NT_{IEP}) • Schlüsselinformation (IK_{IEP}) • Signatur S_1 • Returncode (CC_{IEP})

Tabelle 7.72 Die Funktionalität von DEBIT nach EN 1546-3

DEBIT IEP	
Kommando	• PSAM-Bezeichner (PSAM) • Transaktionsnummer des PSAM (NT_{PSAM}) • abzubuchender Betrag (M_{PDA}) • Währungscode ($CURR_{PDA}$) • Schlüsselinformation (IK_{PSAM}) • Signatur S_2 • benutzerbestimmte Daten (DD)
Antwort	• Signatur S_3 • Returncode (CC_{IEP})

Die eigentliche Transaktion für das Bezahlen führt das folgende Kommando, DEBIT IEP genannt, aus. Auf diese Weise erhält die elektronische Geldbörse in der Chipkarte Informationen über den abzubuchenden Betrag, aktuelle Schlüsselversionen und wiederum eine Signatur. Mit ihrer Hilfe kann, analog zum Aufladen der Börse, die Authentizität des

Sicherheitsmoduls im Terminal geprüft werden. Verläuft die Prüfung erfolgreich, werden das Börsensaldo erhöht, die Protokolldatei für Bezahltransaktionen aktualisiert und eine weitere Signatur S_3 zur Bestätigung des ganzen Vorgangs erstellt. Diese findet sich in der Antwort auf ein DEBIT IEP wieder. Die Signatur S_3 dient dem Sicherheitsmodul im Terminal als Bestätigung, dass der Betrag in der Chipkarte ordnungsgemäß abgebucht worden ist.

Es konnte hier nur ein kurzer Überblick über die vier wichtigsten Kommandos für elektronische Geldbörsen nach EN 1546 gegeben werden. Die entsprechende Norm ist äußerst umfangreich und schließt viele Optionen und Variationsmöglichkeiten für die Systemgestaltung mit ein, welche natürlich auch wieder Auswirkungen auf den Aufbau der Kommandos haben können.⁹

7.15 Kommandos für Kredit- und Debitkarten

Die gemeinsame Spezifikation für Chipkarten im Zahlungsverkehr EMV¹⁰ legt zwei speziell auf den Zahlungsverkehr abgestimmte Kommandos fest. Prinzipiell könnte mit diesen beiden sehr variablen Kommandos sogar eine elektronische Geldbörse auf einer Chipkarte verwirklicht werden. Die Intention liegt jedoch mehr auf dem Einsatz im Umfeld von Kredit- und Debittransaktionen, weshalb hier die Aspekte für diese beiden Anwendungen behandelt werden. Den beiden Kommandos ist ein eigenes Unterkapitel gewidmet, da Kredit- und Debitkarten mit Chip in Zukunft sehr große Stückzahlen erwarten lassen und damit von entsprechend großer Bedeutung sind.

Die beiden Kommandos GET PROCESSING OPTIONS und GENERATE APPLICATION CRYPTOGRAM stützen sich auf TLV-codierte Datenobjekte im Datenteil von Kommando und Antwort. Damit wurde eine erhebliche Vielfalt an möglichen Varianten und Optionen geschaffen, welche von den jeweiligen Anwendungen je nach Bedarf genutzt werden können.

Um eine Zahlung einzuleiten, benutzt man GET PROCESSING OPTIONS. Damit können der Chipkarte TLV-codierte Daten des Terminals für die Bearbeitung des weiteren Zahlungsablaufs übergeben werden (*processing options data object list – PDOL*). Dies kann beispielsweise der Transaktionsbetrag sein. Als Antwort sendet die Karte ein BER-TLV-codiertes Datenobjekt mit dem unterstützten Profil der Anwendung (d. h. jene von der Chipkarte unterstützten Funktionen, die AIP (*application interchange profil*) genannt werden) und dem Ort der Anwendungsdateien (*application file locator – AFL*) zurück.

Tabelle 7.73 Die Funktionalität von GET PROCESSING OPTIONS nach EMV

GET PROCESSING OPTIONS	
Kommando	Liste von Datenobjekten für Transaktionsoptionen (<i>processing options data object list – PDOL</i>)
Antwort	- unterstütztes Profil der Anwendung (<i>application interchange profile – AIP</i>) - Ort der Anwendungsdateien (<i>application file locator – AFL</i>) - Returncode

⁹ Siehe auch Abschnitt 12.3.1 „CEN-Norm EN 1546“.

¹⁰ Siehe auch Abschnitt 12.4 „EMV-Anwendung“.

Das zweite Kommando für den Zahlungsablauf bei einer Kreditkarte mit Chip hat die Bezeichnung GENERATE APPLICATION CRYPTOGRAM. Bei diesem Kommando sind sowohl für die Kommando-APDU als auch für die Antwort-APDU alle Daten TLV-codiert. Im Kommando werden alle notwendigen Daten für eine Zahlungstransaktion sowie das gewünschte Anwendungszertifikat zur Chipkarte gesandt. Die Chipkarte berechnet nun abhängig von übergebenen und gespeicherten Daten den für die Zahlung notwendigen weiteren Ablauf der Transaktion. Als Ergebnis gibt sie ein Anwendungszertifikat an das Terminal zurück, welches im einfachsten Fall das Transaktionszertifikat sein kann. Damit ist dann der Zahlungsablauf beendet.

Das von der Chipkarte zurückgesandte Anwendungszertifikat kann statt des Transaktionszertifikats aber auch eine Autorisierungsanfrage enthalten. Kommt die Chipkarte bei der Ermittlung des weiteren Transaktionsablaufs zu dem Ergebnis, dass eine Online-Autorisierung erforderlich ist, enthält das Anwendungszertifikat in den Antwortdaten eine Anfrage an das dem Terminal übergeordnete Autorisierungszentrum. Nach Bearbeitung durch dieses erhält die Chipkarte in einem zweiten GENERATE APPLICATION CRYPTOGRAM die entsprechenden Daten und kann dann abschließend das Transaktionszertifikat für die Zahlung erstellen und an das Terminal senden.¹¹

Tabelle 7.74 Die Funktionalität von GENERATE APPLICATION CRYPTOGRAM nach dem Industriestandard EMV

GENERATE APPLICATION CRYPTOGRAM	
Kommando	• gewünschtes Anwendungszertifikat • transaktionsbezogene Daten (<i>transaction related data</i>)
Antwort	• Informationen über das Anwendungszertifikat (<i>cryptogram information data</i>) • Transaktionszähler der Anwendung (<i>application transaction counter – ATC</i>) • Anwendungszertifikat (<i>application cryptogram – AC</i>) • Returncode

7.16 Anwendungsspezifische Kommandos

Kommandos, die für eine bestimmte Anwendung maßgeschneidert sind, gibt es viele. Die Gründe dafür liegen meist in der Minimierung des Speicherplatzes oder der Ausführungszeit. Diese Kommandos sind zum Großteil so spezifisch, dass sie keinen Eingang in die Normung finden, oder sie sind durch die Normung auf ein bestimmtes Anwendungsgebiet festgelegt.

Alle anwendungsspezifischen Kommandos hier aufzuzählen, würde den Umfang dieses Kapitels sprengen. Stellvertretend dafür ist das einzige anwendungsspezifische Kommando von GSM 11.11 dargestellt. Es lautet RUN GSM ALGORITHM und wird zur gleichzeitigen Erzeugung eines dynamischen, kartenindividuellen Schlüssels und zur Authentisierung der Chipkarte gegenüber dem GSM-Hintergrundsystem benutzt. Die Funktion ist so spezi-

¹¹ Siehe auch Abschnitt 12.4 „EMV-Anwendung“.

fisch für die Anwendung GSM, dass es nicht sinnvoll wäre, das Kommando in eine allgemeine Norm für Chipkarten aufzunehmen. Das Kommando benutzt einen nur bei GSM verwendeten kryptografischen Algorithmus, und die beiden aus der übergebenen Zufallszahl erzeugten Ausgangswerte wären in allen anderen Anwendungen nutzlos.

Tabelle 7.75 Die Funktionalität von RUN GSM ALGORITHM

RUN GSM ALGORITHM	
Kommando	• Zufallszahl
Antwort	• dynamischer Schlüssel • enc (Schlüssel; Zufallszahl) • Returncode

8 Sicherheitstechnik

8.1	Benutzeridentifizierung	501
8.1.1	Prüfung einer Geheimzahl	503
8.1.2	Biometrische Verfahren	509
8.2	Sicherheit einer Chipkarte	521
8.2.1	Systematik der Angriffe und Angreifer	522
8.2.2	Angriffe und Abwehrmaßnahmen während der Entwicklung	529
8.2.3	Angriffe und Abwehrmaßnahmen während der Produktion	532
8.2.4	Angriffe und Abwehrmaßnahmen während der Kartenbenutzung	533

Die Möglichkeit des Schutzes und der Geheimhaltung von Daten in Chipkarten ist einer der Hauptvorteile gegenüber allen anderen Datenträgern, wie Magnetstreifenkarten oder Disketten. Deshalb ist auch eine auf diesen Zweck abgestimmte und optimierte Chiphardware mit dazu passenden kryptografischen Verfahren zur Sicherung der geheimen Daten unabdingbar. Doch ist Sicherheit nicht alleine von der Spezialhardware des Mikrocontrollers oder den in der Betriebssystemsoftware realisierten kryptografischen Algorithmen abhängig. Auch die Sicherheit der Chipkarten-Anwendung und die Design-Prinzipien, die Entwickler auf dem Weg dorthin anwenden, sind von elementarer Bedeutung. Die wesentlichen Prinzipien, Verfahren und Strategien auf dem Weg zu sicheren Chipkarten und sicheren Chipkarten-Anwendungen sind in diesem Kapitel zusammengefasst.

8.1 Benutzeridentifizierung

Schon seit dem Altertum beschäftigen sich Menschen mit verschiedenen Verfahren zur eindeutigen Identifizierung von Personen. Die einfachste Form der Identifizierung ist die Benutzung eines Ausweises mit Foto oder eine Unterschrift in Gegenwart des Prüfers. Das Foto auf dem Ausweis kann dann mit der echten Person verglichen werden, und das Ergebnis ist eine Aussage über die Echtheit der Person.

Im informationstechnischen Bereich ist dieser Vergleich aber nicht so einfach, da ihn nicht Menschen, sondern Computer vornehmen müssen. Da Computer, trotz aller Erfolge bei stupiden Arbeiten, immer noch große Probleme mit intelligenten Tätigkeiten haben, hat sich die Eingabe von Passwörtern über die Tastatur weitgehend durchgesetzt. Der Aufwand für den Vergleich ist gering, da der Computer im Grunde genommen das übergebene Passwort mit einem gespeicherten Referenzwert vergleicht und dann eine reine Ja-Nein-Entscheidung trifft. Beim Vergleich von Passwörtern ist dies die Entscheidung über Echtheit oder Falschheit der zu überprüfenden Person.

Grundsätzlich gibt es jedoch drei verschiedene Möglichkeiten für die Identifizierung einer Person. Im Falle eines Passwortes prüft man, ob diese Person ein Geheimnis kennt. Ist dies der Fall, schließt man daraus, dass die Person diejenige ist, für die sie sich ausgibt. Analog verläuft die Prüfung, ob die Person im Besitz einer bestimmten Sache ist. Die dritte Möglichkeit besteht darin, bestimmte eindeutige Körpermerkmale dieser Person zu prüfen.

Das Wissen um ein Geheimnis und auch der Besitz einer Sache haben einen bedeutenden Nachteil. Die zu identifizierende Person muss sich etwas merken oder eine Sache mit-

führen. Je nach Situation als Nachteil oder Vorteil zu werten ist, dass bei beiden Methoden eine Übertragung auf andere Personen möglich ist. Damit ist aber nicht mehr eindeutig sichergestellt, dass es auch wirklich der richtige Benutzer ist und nicht jemand, der beispielsweise das Kennwort unrechtmäßig erfahren hat.

Die dritte Variante der Identifizierung schließt eine Übertragbarkeit aus, da sie bestimmte Eigenschaften des menschlichen Körpers zur Identifizierung benutzt. Allerdings ist die auszuführende Messung in den meisten Fällen technisch aufwendig, da man verständlicherweise nicht messtechnisch einfach zu erfassende biologische Merkmale, wie Gewicht oder Körpergröße, verwenden kann.

Einleuchtender und fassbarer werden diese drei Identifizierungsmöglichkeiten, wenn man sich Folgendes durch den Kopf gehen lässt: Man soll sich mit einem unbekannten Menschen am Bahnhof treffen. Sobald man dort jemand passenden sieht, stellt sich die Frage, ob derjenige auch „der Richtige“ ist. Befindet sich der Unbekannte am richtigen Treffpunkt zur richtigen Zeit, wäre dies eigentlich schon die implizite Prüfung eines Geheimnisses, da Zeit und Ort des Treffens (hoffentlich) nicht allgemein bekannt sind. Eine explizite Prüfung eines Geheimnisses bestünde dann darin, dass der Unbekannte ein Kennwort nennt, das nur er und wir kennen. Er könnte sich aber auch durch den Besitz einer Sache ausweisen, beispielsweise durch eine unter den Arm geklemmte Zeitung eines bestimmten Ausgabedatums. Die wohl sicherste Möglichkeit wäre es, wenn man an dem Unbekannten ein bestimmtes körperliches Merkmal prüft. Vielleicht besitzt er eine besonders lange Nase – so wie Pinocchio, wenn er lügt.

Man sieht an dem Bahnhof-Szenario deutlich, dass die Identifizierung eines unbekannten Menschen ein beinahe klassisches Problem sowohl aller Spionageromane als auch des täglichen Leben ist und sich nicht nur auf Computer oder Chipkarten beschränkt.

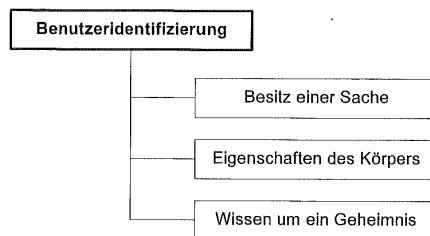


Bild 8.1 Klassifizierungsbaum der Identifizierung einer Person.

An vielen Automaten, Geräten und Computern ist mittlerweile die Eingabe einer PIN selbstverständlich geworden. Durch die damit verbundene starke Zunahme von PINs für die unterschiedlichsten Zwecke ist es für einen Normalbürger schon sehr schwierig geworden, den Überblick zu behalten. Wer kann sich zwanzig (oder mehr) verschiedene PINs merken? Auch ist es der Sicherheit und dem Ruf eines Systems naturgemäß abträglich, wenn jeder Benutzer seine PIN auf die Karte notiert, da die Anzahl der Betrugsfälle überhand nehmen. Aus diesen Gründen ist in den vergangenen Jahren der Wunsch nach anderen Identifikationsverfahren als die PIN-Prüfung laut geworden. Gut dafür geeignet sind so genannte biometrische Merkmale, anhand derer eine Person von einer Maschine eindeutig identifiziert werden kann.

8.1.1 Prüfung einer Geheimzahl

Die häufigste Benutzeridentifizierung ist die Eingabe einer Geheimzahl, welche im Allgemeinen meist als PIN (*personal identification number*) oder manchmal auch als CHV (*card holder verification*) abgekürzt wird.

Üblicherweise besteht die PIN aus vier Stellen, die sich gewöhnlich aus den Dezimalzahlen von "0" bis "9" zusammensetzen. Der Grund für die reine Zahleneingabe ist trivial. Kartenterminals verfügen nämlich in der Regel nur über eine numerische Tastatur. Prinzipiell könnten jedoch natürlich auch alphanumerische Zeichen verwendet werden. Die PIN wird auf der Tastatur eines Terminals oder Computers eingegeben und dann zur Chipkarte gesendet. Diese vergleicht den übergebenen Wert mit einem gespeicherten Referenzwert und teilt dann das Ergebnis dem Terminal mit.

Die Eingabe einer PIN wird vor allem bei Anwendungen im Zahlungsverkehr als sicherheitsrelevant eingestuft, weshalb es in diesem Bereich oft Vorschriften über die Art der Tastatur gibt. Diese besonderen Tastaturen werden oft PIN-Pad genannt. In Deutschland gibt es beispielsweise die Vorschrift (vom ZKA), dass die PIN einer ec-Karte nur an mechanisch und kryptografisch besonders gesicherten Tastaturen eingegeben werden darf. Diese PIN-Pads weisen alle Merkmale eines Sicherheitsmoduls auf (Öffnungsdetektor, Bohrschutzfolie, ...) und verschlüsseln die PIN unmittelbar bei der Eingabe. Damit wird zuverlässig verhindert, dass ein Angreifer diese Tastatur manipulieren und die PIN während der Eingabe abhören kann.

Bei den PINs kann zwischen statischen und veränderbaren PINs unterschieden werden. Eine statische PIN ist vom Benutzer nicht mehr veränderbar und muss von diesem sozusagen auswendig gelernt werden. Ist sie bekannt geworden, müsste der Kartenbenutzer konsequenterweise seine Karte zerstören und sich eine neue Karte mit einer anderen statischen PIN besorgen. Eine veränderbare PIN kann vom Benutzer nach Gutdünken oder auf einen für ihn mnemotechnisch optimalen Wert geändert werden. Sie birgt jedoch eine Gefahr in sich, weil der mnemotechnisch optimale Wert bei vielen Menschen "1234", "4711" und "0815" beträgt. Eine Prüfung auf solche Trivialwerte findet in Chipkarten nicht statt, da der dafür benötigte Speicher in Form einer Tabelle nicht zur Verfügung steht. Es wäre aber durchaus denkbar, dass bereits das Terminal die Änderung auf solch eine PIN verbietet. Zum Ändern der PIN ist es aus Sicherheitsgründen immer notwendig, die aktuell gültige PIN mit zu übergeben, da sonst jede bestehende PIN durch einen Angreifer mit seiner eigenen ersetzt wird.

Anders verhält es sich mit den so genannten Super-PINs oder PUKs (*personal unblocking key*). Diese haben in der Regel mehr Stellen als die eigentliche PIN – ein üblicher Wert ist 6 – und werden dazu benutzt, einen auf seinen Maximalwert stehenden Fehlbedienungszähler einer PIN wieder auf null zurückzusetzen. Mit der PUK wird auch gleich eine neue PIN an die Chipkarte übergeben, da ein zurückgesetzter Fehlbedienungs-zähler wenig nützt, wenn man die PIN vergessen hat. Und dies ist ja meist der Fall, wenn der Fehlbedienungs-zähler seinen Maximalwert erreicht hat.

Es gibt auch Anwendungen, die Transport-PINs verwenden. Die Chipkarte wird mit einer zufälligen PIN personalisiert, welche der Kartenbenutzer in einem PIN-Brief erhält. Bei der ersten Eingabe wird er aber die personalisierte PIN durch seine eigene ersetzen. Bei einem ähnlichen Verfahren, „Nullpin-Verfahren“ genannt, wird die Karte mit einer

Trivial-PIN, wie etwa "0000" vorbelegt, und es wird ebenfalls von der Chipkarte bei der ersten Benutzung ein Wechsel erzwungen. Beide Prinzipien verhindern das Problem, dass eine während der Personalisierung oder Kartenhandhabung ausgespähte PIN zu einem späteren Zeitpunkt noch sinnvoll eingesetzt werden kann.

Nach einer Empfehlung von ISO 9564-1 sollte die PIN eine Länge zwischen vier und zwölf alphanumerischen Zeichen haben, damit die Wahrscheinlichkeit gering ist, die richtige PIN durch bloßes Raten und Probieren herauszufinden. Allerdings sieht die Realität oft etwas anders aus. Die Eingabe von nichtnumerischen Zeichen ist an vielen Stellen aus technischen Gründen überhaupt nicht möglich, da oft nur eine numerische Tastatur vorhanden ist.

Die Anzahl der Stellen einer PIN ist nicht nur von der gewünschten Sicherheit abhängig, sondern sehr stark vom Gedächtnis des durchschnittlichen Kartenbenutzers. Dieser ist aber seit Jahren gewohnt, 4-stellige PINs zu verwenden, weshalb eine Umstellung auf 6-stellige oder noch längere PINs sehr schwierig ist. Auch kann es in der Praxis durchaus vorkommen, dass die vermeintlich sichere 6- oder 8-stellige PIN nur mathematisch besser ist als eine 4-stellige. Viele Menschen haben Probleme, sich so lange Zahlen zu merken, vor allem wenn sie diese selten benutzen, und notieren sie deshalb auf der Karte oder einem Zettel in der Nähe der Karte. Die damit erreichte Sicherheit mit einer längeren PIN ist dann wesentlich schlechter als bei einer kurzen PIN.

Ähnlich verhält es sich mit der durchaus verständlichen Forderung, die PIN periodisch zu wechseln. Für Hochsicherheitsanwendungen mit wenigen Benutzern mag dies funktionieren, bei Massenanwendungen ruiniert dies jedoch die Akzeptanz, da dort möglichst einfache Verfahren verlangt werden, mit denen auch Menschen ohne gutes Gedächtnis problemlos zurechtkommen.

In diesem Zusammenhang ist ein weiterer Punkt von großer Bedeutung. In vielen Fällen ist die Eingabe und Überprüfung der PIN nicht bloß eine Identifikation des Benutzers und beweist den rechtmäßigen Besitz der jeweiligen Karte, sondern stellt gleichzeitig auch eine Willenserklärung dar. Dieser erklärt sich mit Eingabe seiner PIN mit einem bestimmten Vorgang einverstanden. Als Beispiel kann man hier die Eingabe der PIN an einem Geldausgabeautomaten anführen. Zum einen identifiziert sich der Benutzer durch die Kenntnis der geheimen PIN. Zum anderen ist sie aber auch die Willenserklärung des Benutzers, dass er damit einverstanden ist, einen bestimmten Betrag ausbezahlt zu bekommen. Dies ist sehr wichtig im Zusammenhang mit bestimmten biometrischen Merkmalen, bei denen manche nicht unbedingt eine Willenserklärung sein müssen, da sie auch ohne explizite Zustimmung der Person geprüft werden können.

Wahrscheinlichkeit des Erratens einer PIN

Der einfachste Angriff auf eine PIN neben der Beobachtung bei der Eingabe ist das bloße Raten. Wie hoch die Wahrscheinlichkeit ist, ob man dabei Glück hat oder nicht, hängt zum einen davon ab, welche Länge die zu erratende PIN hat, aus welchen Zeichen sie zusammengesetzt ist und wie viele Versuche erlaubt sind. Die Wahrscheinlichkeit, bei dreimaligem Versuch eine vierstellige PIN zu erraten, beträgt 0,03 %, was nicht gerade viel ist. Im Folgenden sind die beiden grundlegenden Formeln für das Erraten von Passwörtern ausgeführt. Sie können praxisnah verwendet werden, um das Risiko bei der Festlegung von Passwortlängen abzuschätzen.

$$x = m^n$$

8.1

$$P = \frac{i}{m^n}$$

8.2

Tabelle 8.1 Definition und Beschreibung der Variablen für die Formeln 8.1 und 8.2

Variable	Beispiel	Beschreibung
<i>i</i>	3	Anzahl der Rateversuche
<i>m</i>	10	Anzahl der möglichen Zeichen pro Stelle
<i>n</i>	4	Anzahl der Stellen
<i>P</i>	0,000 3 (= 0,03 %)	Wahrscheinlichkeit des Erratens des Passwortes
<i>x</i>	10 000	Anzahl der möglichen Passwörter

Es gibt im Übrigen noch ein viertes Kriterium beim Raten einer PIN, das lange Zeit sträflich vernachlässigt wurde. Es ist dies die Gleichverteilung der PINs einer Anwendung. Es ist nämlich sehr viel einfacher, eine PIN zu raten, wenn man weiß, dass bestimmte PINs häufiger vorkommen als andere. Dass diese wichtige Nebenbedingung tatsächlich von Belang ist, wurde im Jahr 1997 plötzlich bei den deutschen ec-Karten relevant. Obwohl das Verfahren, wie eine PIN aus den Daten des Magnetstreifens der ec-Karte errechnet wird, in seinen Details immer noch geheim ist, wurden zumindest einige generelle Schritte des Algorithmus bekannt. Dabei stellte sich heraus, dass die erzeugten PINs nicht gleichverteilt sind, da der verwendete Algorithmus die Zahlen null bis fünf wesentlich häufiger generiert als die Zahlen sechs bis neun. Es wurde auch noch bekannt, dass der Berechnungsalgorithmus eine führende Null bei der PIN-Erzeugung unterdrückt. Das Ergebnis dieser Ungleichverteilung war, dass man nicht mehr 3333 Versuche benötigt, um bei drei erlaubten Fehlversuchen eine 4-stellige PIN zu erraten, sondern nur noch 150 [Karten 97]. Bei 10,5 % der Karten ist die Verteilung sogar so schlecht, dass bereits 72 Versuche ausreichen [Schindler 97], wenn man die Eigenschaften des Algorithmus zur PIN-Generierung berücksichtigt. Das Ergebnis war, dass bei den neuen ec-Karten ein verbesserter Algorithmus zur PIN-Generierung verwendet wird und der ursprünglich benutzte DES auch durch den Triple-DES ersetzt wurde.

Tabelle 8.2 Aufstellung von PINs und Passwörtern mit unterschiedlichen Längen, Codierungen und der Anzahl der möglichen Kombinationen

Art der PIN/des Passwortes	Wertebereich bzw. Codierung der PIN/des Passwortes	Anzahl der möglichen PINs/Passwörter
1-stellige PIN	PIN ∈ {0 ... 9}	10
1-stelliges Passwort	Passwort ∈ {0 ... 9, "A" ... "Z"}	36
1-stelliges Passwort	Passwort ∈ {0 ... 9, "a" ... "z", "A" ... "Z"}	62
1-stelliges Passwort	Passwort ∈ {0 ... 9, "a" ... "z", "A" ... "Z", 20 beliebige Sonderzeichen}	82
4-stellige PIN, keine führende Null	PIN ∈ {1 000 ... 9 999}	9,00 · 10 ³
4-stellige PIN	PIN ∈ {0 000 ... 9 999}	1,00 · 10 ⁴

Tabelle 8.2 (Fortsetzung)

Art der PIN/des Passworts	Wertebereich bzw. Codierung der PIN/des Passworts	Anzahl der möglichen PINs/Passwörter
4-stelliges Passwort	Passwort $\in \{0 \dots 9, "A" \dots "Z"\}$	$1,68 \cdot 10^6$
4-stelliges Passwort	Passwort $\in \{0 \dots 9, "a" \dots "z", "A" \dots "Z"\}$	$1,48 \cdot 10^7$
4-stelliges Passwort	Passwort $\in \{0 \dots 9, "a" \dots "z", "A" \dots "Z", 20 \text{ beliebige Sonderzeichen}\}$	$4,52 \cdot 10^7$
5-stellige PIN, keine führende Null	$PIN \in \{10\,000 \dots 99\,999\}$	$8,9 \cdot 10^4$
5-stellige PIN	$PIN \in \{00\,000 \dots 99\,999\}$	$1,00 \cdot 10^5$
6-stellige PIN, keine führende Null	$PIN \in \{100\,000 \dots 999\,999\}$	$8,99 \cdot 10^5$
6-stellige PIN	$PIN \in \{000\,000 \dots 999\,999\}$	$1,00 \cdot 10^6$
6-stelliges Passwort	Passwort $\in \{0 \dots 9, "A" \dots "Z"\}$	$2,18 \cdot 10^9$
6-stelliges Passwort	Passwort $\in \{0 \dots 9, "a" \dots "z", "A" \dots "Z"\}$	$5,68 \cdot 10^{10}$
6-stelliges Passwort	Passwort $\in \{0 \dots 9, "a" \dots "z", "A" \dots "Z", 20 \text{ beliebige Sonderzeichen}\}$	$3,04 \cdot 10^{11}$

Generierung einer PIN

Um die PIN für eine Chipkarte zu erzeugen, benötigt man nur einen Zufallszahlengenerator und einen Algorithmus, der die Zufallszahl in eine ASCII-codierte PIN der benötigten Länge umwandelt. Nun könnte man noch über eine Tabelle die bekannten Trivalkombinationen erkennen und ausschließen. Anschließend wird die PIN in der Chipkarte gespeichert und mit dem Kommando VERIFY bei Bedarf mit einer übergebenen PIN verglichen.

Werden jedoch in einem System Magnetstreifenkarten verwendet, dann wird die Generierung einer PIN etwas komplizierter. Schließlich muss es möglich sein, dass ein offline arbeitender Geldausgabeautomat anhand der Magnetstreifendaten eine eingegebene PIN prüft. Chipkarten wären davon eigentlich nicht betroffen, doch haben momentan noch alle Debitkarten (z. B.: ec-Karte) aus Kompatibilitätsgründen einen Magnetstreifen, selbst wenn sie mit einem Mikrocontroller versehen sind. Der Algorithmus für die Erzeugung einer PIN muss deshalb bei Hybridkarten mit Chip und Magnetstreifen deterministisch sein, also bei gleichen Eingabewerten immer dasselbe Ergebnis liefern. Ein Zufallszahlengenerator täte dies dagegen nicht.

Es ist also ein Verfahren vonnöten, das anhand von Magnetstreifendaten eine PIN berechnet. Damit die Sicherheit nicht an dem Verfahren als solchem hängt, sollte auch ein geheimer Schlüssel in die Berechnung einfließen.

In Bild 8.2 ist ein Algorithmus dargestellt, der in ähnlicher Form auch bei den ec-Karten in Deutschland benutzt wird. Als Eingabe fungieren Bankleitzahl, Kontonummer und Kartenfolgenummer. Mit dem DES und einem geheimen Schlüssel erzeugt der Algorithmus daraus eine 4-stellige PIN.

Dieses Verfahren hat aber den weiter oben erwähnten Nachteil, dass die erzeugten PINs nicht über den gesamten möglichen Zahlenraum (bei vier Stellen: "0000" ... "9999") gleichverteilt sind. Der Grund dafür ist die Abbildungsvorschrift nach der Verschlüsselung, die alle hexadezimalen Zahlen ('A', 'B', 'C', 'D', 'E', 'F') in Dezimalzahlen umwandelt. Diese unerwünschte Besonderheit könnte aber durch eine verbesserte Abbildungsvorschrift ohne weiteres umgangen werden. Der DES wird zum einen benutzt, damit nicht das Ver-

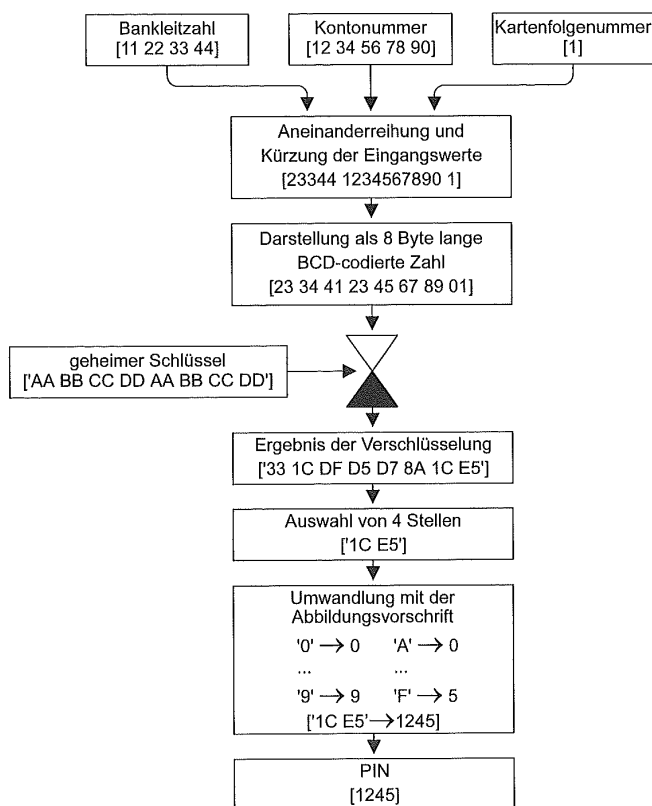


Bild 8.2 Beispiel für die Generierung einer 4-stelligen PIN mithilfe des DES-Algorithmus und der drei kartenindividuellen Daten Bankleitzahl, Kontonummer und Kartenfolgenummer. Das dargestellte Verfahren weist jedoch den Nachteil auf, dass die benutzte Abbildungsvorschrift zu einer Ungleichverteilung der PIN führt (d.h. es gibt PINs, die öfter vorkommen als andere). Die eckigen Klammern umschließen Beispielwerte. Der Ablauf ist dem für die deutsche ec-Karte benutzten Verfahren entfernt ähnlich und basiert auf zwei Beiträgen aus „Die Datenschleuder“ [Müller-Maguhn 97a, Müller-Maguhn 97b].

fahren, sondern nur der Schlüssel geheim gehalten werden muss und zum anderen aufgrund seiner Eigenschaft der Konfusion und Diffusion.¹

Bei deutschen ec-Karten wurde von 1981 bis 1997² ein Verfahren zur Erzeugung der PINs eingesetzt, bei dem diese nicht gleichmäßig über den gesamten Zahlenraum verteilt waren. Dies führte dazu, dass einige PINs deutlich wahrscheinlicher waren als andere, und konnte für (in der Regel nicht erfolgreiche) Angriffe genutzt werden. Es ist deshalb bei allen diesbezüglichen Generierungsverfahren darauf zu achten, dass die erzeugten PINs über den gesamten möglichen Zahlenraum gleichverteilt sind.

¹ Siehe auch Abschnitt 4.6.1 „Symmetrische Kryptoalgorithmen“.

² Siehe auch [Kuhn 97].

Prüfung auf Echtheit des Terminals

Durch die Eingabe der PIN wird bekanntermaßen die Identität des Benutzers überprüft. Dieser möchte aber genauso gerne die Echtheit des Terminals prüfen können. Man denke dabei nur an Dummy-Geldausgabeautomaten. Betrüger konnten so die geheime PIN der Benutzer ermitteln, da sie diese arglos auf der Tastatur des gefälschten Geldausgabeautomaten eingaben. Anschließend stahlen die Betrüger die Karte und konnten mit der ausgespählten PIN Geld abheben. Passieren konnte dies nur, weil es für den Kartenbesitzer keine Möglichkeit gab, die Echtheit des Geldausgabeautomaten zu prüfen.

Gleichwohl gibt es einfache Verfahren, um den vorhergehenden Angriff abzuwehren. In einer Datei in der Chipkarte speichert man ein Kennwort, das nur dem Kartenbenutzer bekannt ist und auch nur von ihm geändert werden kann. Die Datei wird vom Chipkarten-Betriebssystem nur dann für einen Lesezugriff freigegeben, wenn das Terminal durch die Chipkarte authentisiert ist. Als Kennwort kann beispielsweise ein Name oder eine Zahl vom Benutzer vergeben werden.³

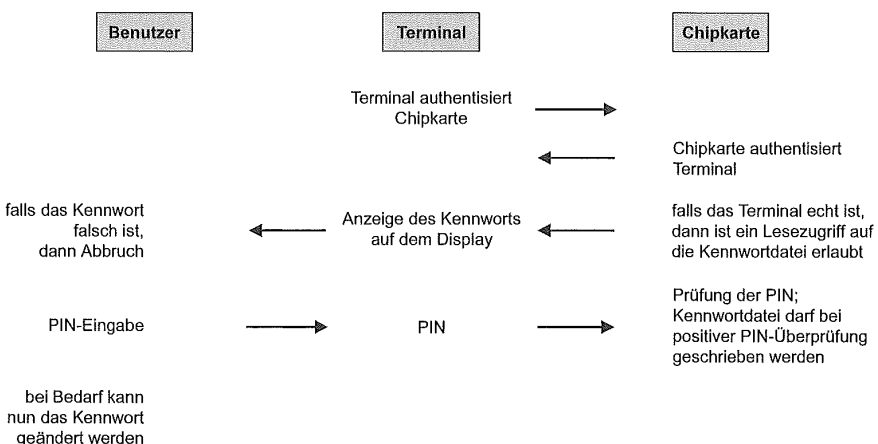


Bild 8.3 Verfahren zur Sicherstellung, dass eine PIN nur an einem echten Terminal eingegeben wird. Voraussetzung dafür ist allerdings, dass der Benutzer seine PIN nicht eher eingibt, als das Kennwort auf dem Display des Terminals korrekt erscheint.

Nachdem der Benutzer seine Chipkarte in ein Terminal gesteckt hat, wird als Erstes eine gegenseitige Authentisierung zwischen Terminal und Karte ausgeführt. Verläuft diese positiv, wissen die beiden Kommunikationspartner, dass das jeweilige Gegenüber echt ist. Nun erlaubt die Karte den Lesezugriff auf die Datei mit dem Geheimnis des Benutzers, und dieses wird auf dem Display des Terminals angezeigt. Der Benutzer sieht nun sein Geheimnis und weiß damit, dass das Terminal echt ist, da er sonst keinen Zugriff auf die Datei mit seinem Geheimnis hätte. Nun kann er unbesorgt seine PIN in das Terminal eingeben.

Mit diesem Verfahren kann man auf einfache Weise ausschließen, dass man seine PIN in manipulierte Terminals eingibt. Als Geheimnis kann ein beliebiges Wort oder eine Zahl

³ Der Aufbau einer entsprechenden Chipkartenanwendung ist im Abschnitt 14.7.3 „Prüfung auf Echtheit eines Terminals“ im Detail vorgestellt.

verwendet werden. Es sollte sich vom Benutzer beliebig ändern lassen, damit ein potentieller Angreifer keine Chance hat, das Geheimnis auszuspähen. Das Verfahren lässt sich für ähnliche Anforderungen entsprechend abändern oder erweitern.

8.1.2 Biometrische Verfahren

Der immer häufigere Einsatz von Passwörtern und PINs führt bei den Anwendern zu immer größeren Widerstand gegen diese Art der Identifizierung. Es stellt sicher für niemanden ein größeres Problem dar, sich einige häufiger verwendete Zahlen oder Buchstabenkombinationen zu merken. Sobald man aber nur selten von seiner kartenindividuellen PIN Gebrauch macht, weil man beispielsweise nur alle zwei Monate einen Geldausgabeautomaten benutzt, wird es für den Normalbürger schwierig, sich die PIN zu merken. Vor allem, wenn er der Stressvorstellung ausgesetzt ist, dass seine Chipkarte nach dreimaliger Falscheingabe vom Geldausgabeautomaten eingezogen wird.

Dies ist sicherlich einer der Hauptgründe, warum man in vielen Bereichen zunehmend biometrische Verfahren favorisiert. Diese Verfahren sind nicht unbedingt schneller oder sicherer als eine PIN-Eingabe, doch für den Benutzer können sie eine große Vereinfachung bedeuten. Falls sich die Sicherheit im Bereich einer PIN-Prüfung bewegt, wird auch ein Systembetreiber biometrischen Verfahren gegenüber aufgeschlossen sein. Schließlich sind biometrische Merkmale nicht wie eine PIN auf andere Menschen übertragbar. Man identifiziert also die Person selber und nicht ein Geheimnis (die PIN), das eine Person gemeinsam mit dem Systembetreiber hat.

8.1.2.1 Grundlagen

Ein biometrisches Identifizierungsverfahren ist ein Verfahren, das auf der Grundlage von einzigartigen, individuellen und biologischen Merkmalen eine Person eindeutig identifizieren kann. Dabei unterscheidet man zwischen physiologischen und verhaltensbasierten Merkmalen. Sind die durch das biometrische Identifizierungsverfahren geprüften Merkmale direkt mit dem Körper einer Person verbunden und haben keine Abhängigkeit von bewussten Verhaltensmustern, so spricht man von physiologischen biometrischen Merkmalen. Das Gegenteil – die verhaltensbasierten biometrischen Verfahren – benutzen als Grundlage bestimmte Merkmale, die innerhalb gewisser Grenzen bewusst veränderbar, aber doch typisch für eine Person sind.

Gerade im Zusammenhang mit der Prüfung von biometrischen Merkmalen ist ein wesentlicher Aspekt die dazu notwendige Benutzerakzeptanz. Die Benutzer werden ein Verfahren umso mehr akzeptieren und bereit sein, es einzusetzen, je Ähnlicher es bereits vorhandenen und bekannten Identifizierungsverfahren ist. Ein typisches Beispiel wäre die eigenhändige Unterschrift, die seit Generationen zur Identifizierung und Willenserklärung in beinahe allen Kulturkreisen benutzt wird. Auch soziale Aspekte spielen eine große Rolle. Fingerabdruckverfahren werden in vielen Ländern vor allem von Polizei und Sicherheitsbehörden eingesetzt, weshalb dann bei darauf basierenden biometrischen Verfahren unter Umständen die Akzeptanz leidet.

Ein weiterer zu beachtender Punkt sind medizinische und hygienische Bedenken der Benutzer. Das können Ängste vor einer Übertragung von Krankheiten beim optischen Ab-

tasten des Augenhintergrundes sein oder die Befürchtung, Schaden am Auge durch den dabei verwendeten Laser zu erleiden. Selbst wenn diese Ängste völlig subjektiv sind und einer wissenschaftlichen Begründung entbehren, werden sie doch das Verhalten und vor allem die Akzeptanz der Benutzer stark beeinflussen. Vor jedem Einsatz von biometrischen Identifikationsverfahren sollte man sich dessen bewusst sein.

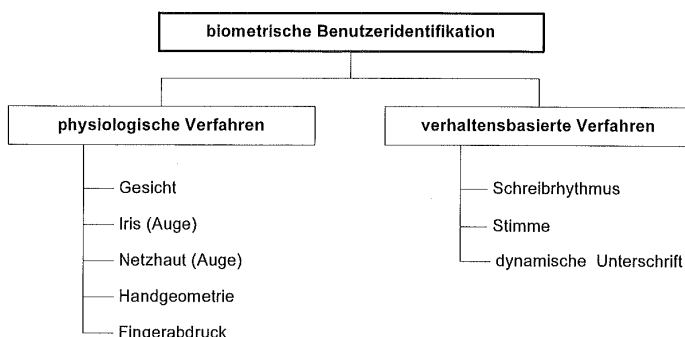


Bild 8.4 Klassifizierungsbaum der wichtigsten biometrischen Verfahren zur Benutzeridentifikation.

Im Vergleich zu wissensbasierten Identifikationsverfahren haben biometrische Verfahren noch eine Eigenschaft, die je nach Standpunkt sowohl von Vorteil als auch von Nachteil sein kann. Es gibt keine Übertragbarkeit bei biometrischen Merkmalen, d. h. bei Systemen, die eine Identifikation mit biometrischen Merkmalen vornehmen, ist es beispielsweise nicht mehr möglich, seine Chipkarte inklusive PIN einer Person seines Vertrauens zu borgen, die dann die Karte entsprechend ihrem Zweck nutzen kann. Anwendungsbetreibern stehen bei diesem Beispielsszenario sicherlich die Haare zu Berge, da bei fast allen Systemen die Weitergabe der PIN untersagt ist. Allerdings hat wohl jeder Leser genügend Erfahrung, um zu wissen, wie leger solche Beschränkungen in der Realität gehandhabt werden.

Biometrische Merkmale sind üblicherweise nicht änderbar – gerade diese Eigenschaft macht diese Merkmale schließlich zur eindeutigen Identifizierung von Personen interessant. Diese Nichtänderbarkeit kann jedoch im Falle der Kompromittierung durchaus zu großen Komplikationen führen. Da sich manche biometrische Merkmale zudem auch noch ohne Einverständnis der jeweiligen Person mit überschaubarem Aufwand aufnehmen lassen, kann dies in Verbindung mit der Nichtänderbarkeit zu einem ernststen Problem werden. An einem Beispiel lässt sich dies gut verdeutlichen: Unberechtigterweise werden die Fingerabdrücke einer Person durch entsprechende Analyse von in die Hand genommenen Gegenständen beim Essen in einem Restaurant aufgenommen. Diese Daten werden anschließend im Internet veröffentlicht, so sich die Fingerabdrücke mit entsprechenden Hilfsmitteln bei Bedarf von jedem nachmachen lassen. Eine eindeutige Identifizierung dieser Person mittels des biometrischen Merkmals Fingerabdruck ist auf Lebenszeit zweifelhaft, da nicht mehr sichergestellt werden kann, dass es sich wirklich um diese Person handelt.

Man kann die Merkmale nach ihrer Ausspähbarkeit in offen, leicht verdeckt, verdeckt und schwer verdeckt einordnen. Je nach Klassifizierung sind sie mehr oder minder einfach

durch Dritte unerlaubt ermittelbar. Offene Merkmale, wie das Gesicht, könnten durch einfache Beobachtung aufgenommen werden. Ein leicht verdecktes Merkmal wäre beispielsweise ein Fingerabdruck, der mit einfachen Hilfsmitteln unbemerkt aufgenommen werden kann. Die Ausspähung der Netzhaut erfordert bereits viel technischen Aufwand und ist unbemerkt annähernd unmöglich, weshalb es zur Kategorie der verdeckten Merkmale gehört. Schwer verdeckte Merkmale sind oftmals verhaltensbasierte Merkmale, da diese vor allem bewusst preisgegeben werden.

Die Eingabe der PIN ist nicht nur die Überprüfung, ob der Benutzer ein Geheimnis kennt, sondern auch die juristische Willensbekundung „ich bin einverstanden“. Dieser Zusammenhang ist sehr wichtig, wenn man andere Verfahren als die PIN-Prüfung verwenden will. Es kann mit Sicherheit keine Willensbekundung eines Benutzers für eine irgendwie geartete Aktion sein, wenn sein Augenhintergrund aus drei Metern Entfernung überprüft wird, was im Übrigen technisch zurzeit ohnehin nicht möglich wäre. In fast allen Ländern muss der Benutzer bewusst etwas manuell ausführen, damit dies als Willensbekundung ausgelegt werden kann. Beispielsweise ist das Brechen eines Siegels auf einer Pappschachtel mit Software eine eindeutige Willensbekundung, dass der Benutzer sich mit den aufgedruckten Lizenzbestimmungen einverstanden erklärt. Biometrische Verfahren, bei denen eine Person bei völliger Passivität geprüft wird, müssen deshalb durch eine entsprechende Benutzerführung zusätzlich mit dem Element der Willensbekundung ergänzt werden.

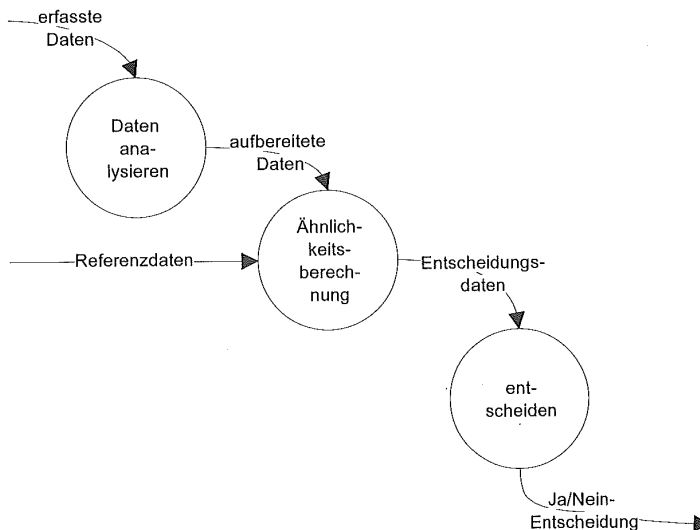


Bild 8.5 Grundsätzlicher Datenfluss bei der rechnerischen Auswertung eines biometrischen Merkmals.

Zur Identifizierung einer Person eignen sich natürlich nicht alle biologischen Merkmale. Die folgenden Punkte müssen zumindest erfüllt sein, damit sich ein Merkmal auch sinnvoll nutzen lässt:

- Das Merkmal muss technisch gut messbar sein (Messmethode, Messdauer, Messkosten).
- Das Merkmal muss eindeutig einer bestimmten Person zuzuordnen sein.
- Das Merkmal muss in der Bevölkerung weit verbreitet sein.
- Eine Veränderung des Merkmals in betrügerischer Absicht darf nicht möglich sein.
- Die erzeugten Referenzdaten müssen klein sein (wenige hundert bis maximal einige tausend Bytes).
- Die natürlichen Veränderungen des Merkmals über die Zeit müssen so gering sein, dass das Merkmal immer einwandfrei messbar bleibt.
- Die Messmethode und das Merkmal müssen von den Benutzern akzeptiert werden.

Bei Messungen aller Art wird man feststellen, dass das Messergebnis nicht immer gleich ist, sondern je nach Messung differiert. Selbst bei sehr einfach durchzuführenden Messungen tritt dieser Effekt auf. Misst man beispielsweise die Länge eines Blattes Papier, so wird sich das Ergebnis bei mehreren Messungen immer etwas unterscheiden. Dies hat viele Ursachen und ist aber nicht weiter problematisch, weil sich der Mittelwert aller Messungen dem exakten Wert annähern wird.

Aus der Erfahrung kann man ableiten, dass die Abweichungen bei einer Messung umso größer sind, je aufwendiger der eigentliche Messvorgang ist. Beispielsweise ist es messtechnisch gesehen ein großer Unterschied, ob man das Gewicht einer Tafel Schokolade oder den Abstand Erde-Mond bestimmt. Messungen an Personen gehören hier auf jeden Fall in die Kategorie schwierig und sind mit großen Streuungen behaftet.

Im Bild 8.6 ist als Beispiel die Messung eines biometrischen Merkmals angegeben. Es könnte beispielsweise die Länge eines Fingers sein. Auf der Abszisse (x-Achse) sind die möglichen Streuungen bei der Messung aufgetragen. Die Ordinate (y-Achse) gibt die Wahrscheinlichkeit der korrekten Identifikation aufgrund des gemessenen biometrischen Merkmals an. Bei einem idealen biometrischen Merkmal und einer idealen Messmethode gäbe es bei den einzelnen Messungen keinerlei Abweichungen. Der Kurvenverlauf würde sich auf eine senkrechte Linie reduzieren. Ein reales Merkmal mit einer realen Messmethode führt aber zu der eingezeichneten Gauß'schen Glockenkurve. Tritt bei der Messung eine Abweichung vom Referenzwert auf, so kann man natürlich nicht mehr mit hundertprozentiger Wahrscheinlichkeit annehmen, dass die zu identifizierende Person richtig erkannt wurde.

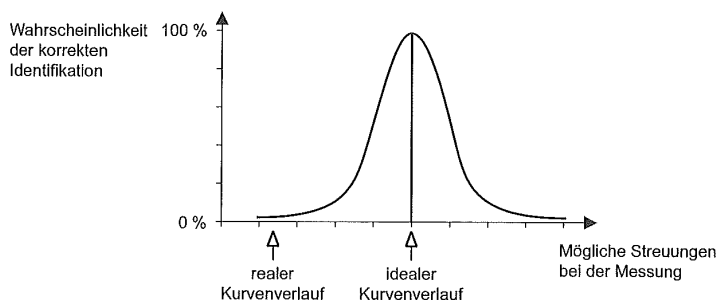


Bild 8.6 Wahrscheinlichkeitsverteilung bei wiederholten Messungen eines biometrischen Merkmals an einer Person.

Um ein biometrisches Merkmal prüfen zu können, muss bei der betreffenden Person vorher das Merkmal erfasst werden. Dies kann durch wiederholte Messung mit Mittelwertbildung geschehen. Das Ergebnis ist ein Referenzwert, der dann in der Chipkarte gespeichert wird. Diese prüft dann bei Bedarf, ob das Referenzmuster mit dem übergebenen aktuellen Messwert übereinstimmt. Je nach biometrischem Verfahren kann es aber notwendig sein, dass die aktuellen Messwerte vorab durch einen leistungsfähigen Computer für den Vergleich in der Chipkarte aufbereitet werden müssen. Da die Identifizierung nicht mit hundertprozentiger Sicherheit erfolgen kann, benötigt man einen Grenzwert, ab dem eine Person als echt erkannt wird. Dieser muss je nach Verfahren und Anwendung festgelegt werden.

Zeichnet man nun in das Diagramm der Wahrscheinlichkeitsverteilung neben der Kurve für die zu identifizierende Person noch eine Kurve für eine weitere Person ein, so erhält man Bild 8.7. Die zusätzliche Kurve steht für eine beliebige Person, bei der die Messkurve für ihr biometrisches Merkmal nahe genug an der Kurve für die zu prüfende Person liegt, damit es Einfluss auf die Entscheidung über die Identität hat. Da sich die beiden Kurven asymptotisch einer Horizontalen annähern, besitzen sie einen gemeinsamen Schnittpunkt. An diesem Punkt ist die Wahrscheinlichkeit, dass es sich um die richtige Person handelt, genauso groß wie dass es sich um eine falsche Person handeln könnte. Deshalb wird bei biometrischen Identifikationssystemen noch ein justierbarer Grenzwert verwendet, mit dem man angibt, ab welcher Wahrscheinlichkeit man von einer korrekten Identifikation ausgeht. Der eingezeichnete Grenzwert teilt die beiden Kurven in vier Bereiche. Die Bereiche geben an, welche Entscheidung über die Identität der Person bei einer Messung des biometrischen Merkmals gefällt wird: Person identifiziert – Person nicht identifiziert.

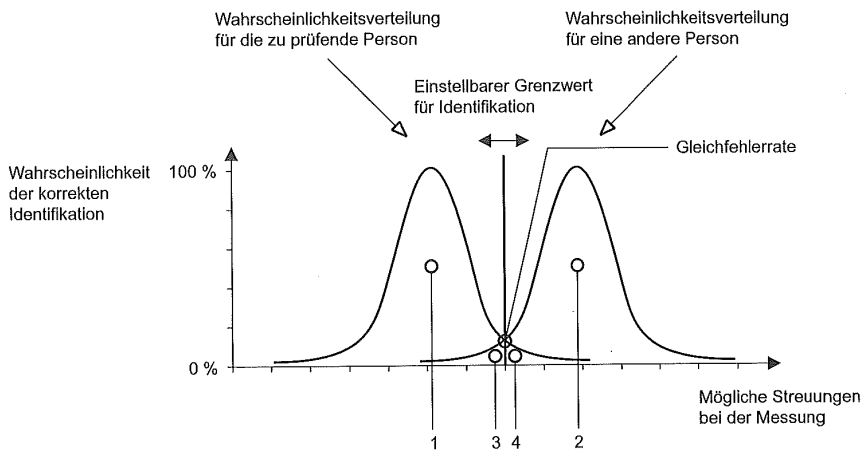


Bild 8.7 Wahrscheinlichkeitsverteilung und Entscheidung bei der Messung eines biometrischen Merkmals.

- 1 – Bereich einer korrekten Identifizierung (*true acceptance*)
- 2 – Bereich einer korrekten Rückweisung (*true rejection*)
- 3 – Bereich einer fehlerhaften Identifizierung (*false acceptance*), Bereich der FAR
- 4 – Bereich einer fehlerhaften Rückweisung (*false rejection*), Bereich der FRR

Als Kernaussage zeigt das Diagramm, dass eine zweifelsfreie Identifikation eines Benutzers nie gemacht werden kann. Es ist lediglich möglich, mit hoher Wahrscheinlichkeit anzunehmen, dass es sich um die richtige Person handelt. Die Größe dieser Wahrscheinlichkeit ist mit dem Grenzwert einstellbar. Allerdings ist die Lage des Grenzwertes in der Praxis nicht ganz beliebig, da ein strenger Maßstab bei der korrekten Identifizierung auch eine große Menge an fehlerhaften Rückweisungen bedeutet.

Die beiden grundlegenden Werte zur Beurteilung eines biometrischen Verfahrens sind FAR (*false acceptance rate*) und FRR (*false rejection rate*). FAR ist die Wahrscheinlichkeit für eine fehlerhafte Akzeptanz der falschen Person und FRR die Wahrscheinlichkeit für eine fehlerhafte Rückweisung einer echten Person. Diese beiden Wahrscheinlichkeitswerte sind natürlich nicht beliebig wählbar, sondern sind vor allem Eigenschaften des jeweils benutzten biometrischen Verfahrens und nur beschränkt innerhalb gewisser Grenzen einstellbar. Des Weiteren bestehen zwischen FAR und FRR Abhängigkeiten in der Form, dass eine niedrige FRR eine höhere FAR verursacht und umgekehrt.

Für den Benutzer bedeutet eine hohe FRR, dass er unter Umständen trotz Legitimation zurückgewiesen wird. Dies beeinträchtigt natürlich die Akzeptanz. Der Systembetreiber ist neben einer niedrigen FRR auch an einer niedrigen FAR interessiert, damit keine fehlerhaften Identifizierungen vorkommen.

Die Prüfung einer PIN setzt in der Chipkarte keine aufwendigen Algorithmen voraus. Es muss lediglich ein Vergleich zwischen der übergebenen und der in der Chipkarte gespeicherten PIN ausgeführt werden. Bei biometrischen Merkmalen ist dies leider nicht so einfach. Der Referenzwert ist zwar in der Chipkarte gespeichert, doch der Vergleich mit den jeweiligen Messergebnissen kann in der Regel nicht in der Chipkarte ausgeführt werden. Die Gründe dafür liegen in der zur Auswertung von biometrischen Merkmalen notwendigen hohen Rechenleistung. Da Chipkarten in der Regel die benötigte Rechenleistung noch nicht zur Verfügung stellen können, werden die rechenintensiven Vorberechnungen der Messwerte extern durchgeführt. Die so gewonnenen Ergebnisse sendet man dann zur Chipkarte, die diese vorverarbeiteten Daten mit speziellen Algorithmen ohne großen Speicher- und Rechenleistungsbedarf weiter verarbeitet und mithilfe des abgelegten Referenzwertes eine Ja/Nein-Entscheidung trifft. Dieses Verfahren bezeichnet man auch als On-Card-Matching oder Matching-On-Chip (MOC). Die Dauer der Prüfung hängt dabei von vielen Parametern ab, so vom verwendeten biometrischen Verfahren und auch einer eventuellen Vorverarbeitung der Daten. So dauert beispielsweise die Prüfung von im Terminal vorverarbeiteten Fingerabdruckdaten auf einer Chipkarte mit 8 Bit Prozessor etwa 2 Sekunden.

Biometrische Merkmale sind persönliche Daten, und deshalb sollten sie auch entsprechend geschützt werden. Dies ist eine sehr gute Anwendung für Chipkarten, da die zur Prüfung benötigten Referenzdaten niemals die Chipkarte verlassen müssen und so auch Angriffe deutlich erschwert werden. Sind hingegen die Referenzdaten in einer unsicheren Umgebung gespeichert, können sie beliebig manipuliert und ausgelesen werden. Damit würde ein biometrisches Identifikationsverfahren keinen wesentlichen Vorteil mehr erbringen. Mit steigender Rechenleistung der Mikrocontroller und der zukünftigen Möglichkeit, den Sensor für die biometrische Datenaufnahme auf der Karte zu integrieren, eröffnet sich damit ein neues weites Einsatzgebiet für Chipkarten.



Bild 8.8 Foto eines USB-Tokens mit integriertem Fingerabdrucksensor und Chipkarten-Mikrocontroller. (Giesecke & Devrient).

8.1.2.2 Physiologische Merkmale

Es wäre in den folgenden Kurzbeschreibungen ohne weiteres möglich, noch zusätzliche biometrische Merkmale aufzuführen, doch aus Gründen der Übersichtlichkeit sind nur die wesentlichen und am häufigsten verwendeten behandelt.

Die nicht bewusst änderbaren physiologischen Merkmale unterliegen zum Teil geringen Veränderungen über die Zeit. So ändern sich die charakteristischen Muster von Fingerabdrücken im gesamten Leben nie, genauso wenig wie die Blutgefäße der Netzhaut. Eine Ausnahme stellt dabei sicherlich das Gesicht dar, das sich natürlich prinzipiell ebenfalls nicht verändert, aber durch Frisur, Bart und Ähnliches sehr stark verwandelt werden kann. Grundsätzlich ist aber trotzdem zu sagen, dass bei biometrischen Merkmalen, die auf der Physiologie von erwachsenen Menschen beruhen, keine laufenden Anpassungen des Referenzmusters notwendig sind, da Änderungen zu gering, beziehungsweise gar nicht vorhanden sind.

Gesicht

Aus der täglichen Erfahrung heraus kann man annehmen, dass sich das menschliche Gesicht ebenfalls gut als biometrisches Merkmal eignet. Die Umsetzung dieser Annahme in die technische Realisierung ist aber mit vielen Tücken verbunden. Gesichter können sich innerhalb kurzer Zeit stark verändern, und ihr Aussehen hängt sehr stark von äußeren Umständen ab. Man denke nur an Brille, Bart, Make-up, Beleuchtung oder Aufnahmewinkel des Gesichtes.

Tastet man ein Gesicht mit einer Kamera im normaloptischen Bereich ab und bereitet die so erzeugten Daten entsprechend auf, kann man in vielen Fällen eine Aussage über die Identität des Menschen hinter dem Gesicht treffen. Dass dazu an technischer Ausrüstung unter anderem sehr leistungsfähige Computer, Fuzzy-Logik und neuronale Netzwerke notwendig sind, verdeutlicht den Aufwand, den man dafür betreiben muss. Zusätzlich sollte das aufgewertete Bild entweder dreidimensional sein oder die zu prüfende Person sich bei der Aufnahme drehen, damit das System nicht betrogen werden kann, indem man ein Foto vor die Kamera hält. Man kann zu diesem Merkmal allgemein sagen, dass es vielleicht in Zukunft eine sehr interessante Methode für ein biometrisches Verfahren wäre, doch momentan noch nicht die Trefferwahrscheinlichkeit aufweist, um es im großen Stil einsetzen zu können.

Netzhaut

Die Netzhaut des menschlichen Auges ist aufgrund ihrer Knoten und Verzweigungen der Blutbahnen für jede Person unterschiedlich. Abgetastet wird mit einem die Pupille durchdringenden Lichtstrahl im infraroten Wellenlängenbereich. Das von der Netzhaut reflektierte Licht wird zurück in eine CCD-Kamera reflektiert, die dann die aufgenommenen Bilddaten an einen Analysecomputer weitergibt.

Die Abtastung der Netzhaut ist eines der besten biometrischen Merkmale überhaupt, da sich mit ihr eine Person mit sehr hoher Wahrscheinlichkeit eindeutig identifizieren lässt. Allerdings stößt das Verfahren bei den Benutzern auf Ablehnung, da man zur Identifizierung seine Augen sehr nahe an das Abtastgerät heranbringen muss. Dadurch kommen oft Ängste vor Ansteckung von Krankheiten und dem abtastenden Infrarotstrahl auf.

Iris

Die Iris (Regenbogenhaut) begrenzt als variable Lochblende den Strahlengang zur Netzhaut. Sie ist ein biologisches Merkmal, das ähnlich wie die Netzhaut für eine bestimmte Person eindeutig ist.

Bei der Abtastung der Iris kann man mehr Abstand von dem messenden Gerät halten als bei der Prüfung der Netzhaut, da das Messverfahren einfacher ist. Bei diesem Verfahren wird die an der Oberfläche des Auges befindliche Iris im normaloptischen Bereich durch eine CCD-Kamera aufgenommen. Die Auswertung folgt in etwa analog der Auswertung der Netzhaut. Es können hier Probleme durch Kontaktlinsen auftreten, da diese die Messung unter Umständen stark beeinflussen.

Geometrie der Hand

Schon seit den 70er-Jahren werden Identifikationssysteme eingesetzt, die auf der Grundlage einer dreidimensionalen Vermessung der Hand oder Teilen davon arbeiten. Als Ausgangspunkt der Messungen können beispielsweise Fingerlänge, Fingerdurchmesser und Radius der Fingerkuppen verwendet werden. Mit nur sehr wenigen Abtastpunkten (z. B.: 5) lässt sich ein für jeden Menschen einzigartiges Merkmal festlegen. Die Abtastung selber kann technisch sehr einfach durch Infrarot-LEDs und Fotodioden geschehen. Die Grundlage ist dabei die vollständige oder teilweise Abdeckung der Infrarotstrahlengänge durch die Geometrie der Hand. Da nur wenige Abtastpunkte für eine Identifizierung genügen, ist das Verfahren auch hinreichend schnell und für den Benutzer unkompliziert. Er legt einfach seine Hand in ein Gerät, und dieses nimmt die Messung vor.

Fingerabdruck

Das bekannteste auf physiologischen Grundlagen beruhende biometrische Verfahren ist sicherlich die Identifizierung mittels Fingerabdruck. In der elektronischen Variante müssen dazu natürlich nicht mehr Daumenabdrücke mit schwarzer Tusche auf Papier gemacht werden. Der Daumen oder auch eine andere Fingerkuppe wird auf eine durchsichtige Platte gelegt, und eine darunter angebrachte Kamera tastet die Hautoberfläche berührungslos ab. Alternativ werden auch Ultraschallsensoren und kapazitive Sensoren auf Halbleiterbasis verwendet.

Für den Vergleich mit dem Referenzmuster benutzt man im Regelfall die Primärmerkmale des Klassifikationsverfahrens nach Edward Richard Henry:⁴ Bogen, Schleife, Wirbel. Man speichert den Typ, die Position und die Ausrichtung von etwa zwanzig dieser Merkmale und erstellt daraus die Datengrundlage für das Referenzmuster. Diese charakteristischen Merkmale bezeichnet man auch als Minutien.

In manchen Benutzerkreisen gibt es eine Abneigung gegen dieses Verfahren, da es seit vielen Jahren in der Verbrechensbekämpfung eingesetzt wird. Problematisch können Verletzungen an den Fingerkuppen für die einwandfreie Personenidentifizierung sein. Ebenfalls gibt es bei diesem Verfahren Identifizierungsprobleme mit Menschen, die viel mit ihren Händen arbeiten. Viele Systeme besitzen zusätzlich zur Abtasteinheit Sensoren zur Messung von Temperatur oder des Pulses im Finger. Damit soll sichergestellt werden, dass keine von der Hand abgetrennten Finger zur Identifizierung verwendet werden können.

Trotzdem sind Fingerabdruck-Systeme sehr weit verbreitet, da sie vom technischen Aufwand und von der Benutzerakzeptanz her verhältnismäßig problemlos sind. Auch bewegt sich die Abtastzeit und die anschließende Prüfdauer noch im angemessenen Rahmen. Die Auflösung bewegt sich dabei in der Größenordnung von 400 dpi.

Tabelle 8.3 Vergleich der biometrischen Verfahren, die auf physiologischen Eigenschaften basieren. Die Werte sind typische Werte und können von Hersteller zu Hersteller stark differieren. Die Größe der Referenzdaten ist zusätzlich sehr stark von der Vorverarbeitung abhängig.

	Dauer der Prüfung	Größe des Referenzwerts	Wahrscheinlichkeit für eine fehlerhafte Rückweisung (FRR)	Wahrscheinlichkeit für eine fehlerhafte Akzeptanz (FAR)
Gesicht	---	---	$\approx 2\%$	$\approx 1\%$
Netzhaut	1,5 – 7 s	40 – 80 Byte	$0,5 \cdot 10^{-2}\%$	$10^{-9}\%$
Geometrie der Hand	1 – 2 s	10 – 30 Byte	0,8 %	0,8 %
Fingerabdruck	1,5 – 9 s	300 – 800 Byte	$1,4 \cdot 10^{-2}\%$	$10^{-6}\%$

8.1.2.3 Verhaltensbasierte Merkmale

Verhaltensbasierte biometrische Merkmale bleiben über die Zeit hinweg bei vielen Personen nicht stabil. Man denke nur an die Unterschrift, die im Laufe des Lebens starken Veränderungen unterworfen ist. Diese Veränderungen treten aber in den seltensten Fällen schlagartig auf, sondern ganz allmählich und langsam. Deshalb benutzten viele Systeme hier adaptive Verfahren, die festgestellte Veränderungen an dem biometrischen Merkmal bei korrekter Identifikation dann als neues Referenzmuster übernehmen und in der Chipkarte speichern.

⁴ Gegen Ende des neunzehnten Jahrhunderts wurden von der bengalischen Polizei in Indien unter Führung des britischen Polizeibeamten Sir Edward Richard Henry erstmalig Fingerabdrücke zur Identifizierung von Personen benutzt. Zurückgekehrt nach London, stellte Edward Henry im Jahr 1901 die erste britische Fingerabdruckkartei zusammen und erstellte die bis in die heutige Zeit gültige Klassifikation der Merkmale eines Fingerabdrucks.

Schreibrhythmus

Man hat festgestellt, dass es beim Eintippen von Zeichen auf einer Tastatur sehr große Unterschiede zwischen einzelnen Personen gibt. Sie liegen vor allem in den Pausen beim Tippen der einzelnen Zeichen. Dies kann man natürlich als biometrisches Merkmal zur Identifizierung benutzen. Das Verfahren funktioniert, indem der zu identifizierende Benutzer eine vorgegebene und mit jeder Prüfung wechselnde Zeichenkette auf der Tastatur eintippt und der daran angeschlossene Computer dazu parallel die Auswertung des Schreibrhythmus vornimmt. Zur Erkennung des Schreibrhythmus lässt sich auch ein beliebiger Text benutzen, doch müssen dann mehr Zeichen eingetippt werden als mit vorgegebenem Text.

Der große Vorteil des Verfahrens ist, dass man keine zusätzliche Hardware benötigt, da eine Tastatur und ein Computer in den meisten Fällen ohnehin bereitsteht. Leider benötigt man bei diesem Verfahren zwischen 100 und 150 alphanumerische Zeichen, welche noch dazu im Zehnfingersystem eingetippt werden müssen, um eine Person sicher zu erkennen. Dies ist der größte Nachteil des Verfahrens.

Stimme

Ähnlich wie das Gesicht ist auch die Stimme für einen bestimmten Menschen charakteristisch. Man kann sie deshalb ebenfalls zur Identifikation von Personen benutzen. Dazu spricht die zu identifizierende Person einen oder mehrere Sätze in ein Mikrofon. Diese müssen bei jeder Identifikation anders sein, da sich sonst sehr einfach ein Angriff durch Wiedereinspielung der bei einer vorherigen Identifikation beispielsweise auf Tonband aufgenommenen Sätze durchführen ließe. Mit den Wellenformen der gesprochenen Sätze führt man nun eine Fourier-Analyse durch, um das für jede Person charakteristische Wellenspektrum herauszufinden. Dieses vergleicht man anschließend mit einem Referenzwert und entscheidet über die Identität des Sprechers. Zusätzlich wird auch noch die ganze Palette an neueren Verfahren der Informationstechnik wie Fuzzy-Logik, neuronale Netze und Ähnliches eingesetzt.

Die Methode ist natürlich nicht ohne Nachteile. Die Stimme eines Menschen ist sehr stark von seiner aktuellen körperlichen Verfassung abhängig. Auch müssen bei einer Stimmerkennung zuverlässig alle Hintergrundgeräusche ausgeblendet werden, um überhaupt eine eindeutige Spektralanalyse durchführen zu können. Um Angriffe durch Wiedereinspielung zu verhindern, muss bei jeder Identifikation ein anderer Satz aufgesagt werden, was das Verfahren sehr erschwert und die Erkennung aufwendig macht. Diesen technischen Schwierigkeiten steht allerdings eine gute Benutzerakzeptanz gegenüber, was diese Methode der biometrischen Benutzeridentifikation durchaus interessant macht.



Bild 8.9 Lautstärkeverlauf und Frequenzspektrum des Namens „Wolfgang“, von zwei verschiedenen Personen gesprochen.

Dynamische Unterschrift

Die einzige im täglichen Umgang gewohnte Identifizierungsmethode ist die Leistung einer Unterschrift. Sie kann aufgrund des sehr individuellen Charakters ebenfalls als biometrisches Merkmal verwendet werden. Bei statischen Verfahren bewertet man bereits geleistete Unterschriften. Bei dynamischen Verfahren misst man hingegen während des Vorgangs des Unterschreibens. Das statische Verfahren ist eher theoretischer Natur, da es fotokopierte Unterschriften nicht von echten unterscheiden kann.

Die bei den dynamischen Verfahren gemessenen Parameter können beispielsweise Gestalt der Unterschrift, Geschwindigkeit, Beschleunigung, Anpresskräfte des Stifts auf den Schreibuntergrund und die Zeitdauer sein. Zur Messung können spezielle Stifte verwendet werden oder eine für die zu messenden Parameter sensitive Schreibunterlage. Im Bild 8.9 ist beispielhaft ein möglicher Aufbau gezeigt, mit dem man mit normalen Stiften auf einer speziellen Unterlage schreibt und die dabei gemessenen Signale als Grundlage für eine biometrische Identifizierung dienen können. An den Kreuzungspunkten der beiden Leitungen ist ein Drucksensor angebracht, dessen Signalamplitude über eine Auswertlogik zu einem Computer gesandt wird. Dieser kann dann mit unterschiedlichen Algorithmen die gemessenen Daten in eine standardisierte Form bringen und mit dem gespeicherten Referenzmuster vergleichen.

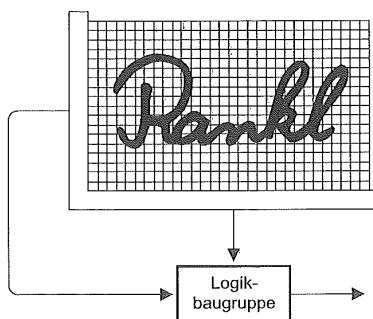


Bild 8.10 Beispiel für eine Messanordnung zur Prüfung einer dynamisch erzeugten Unterschrift.

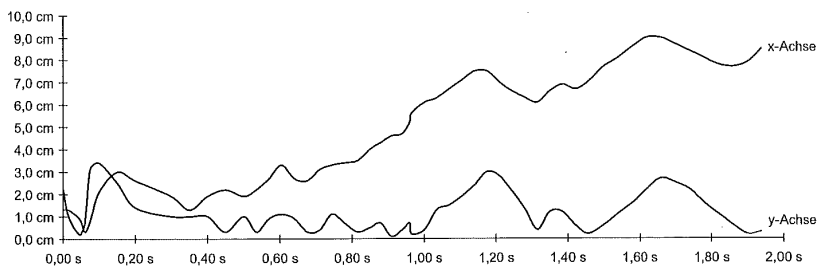


Bild 8.11 Position des Schreibstifts in horizontaler (x-Achse) und vertikaler (y-Achse) Richtung im Verlauf über die Zeit beim Schreiben des Wortes „Rankl“.

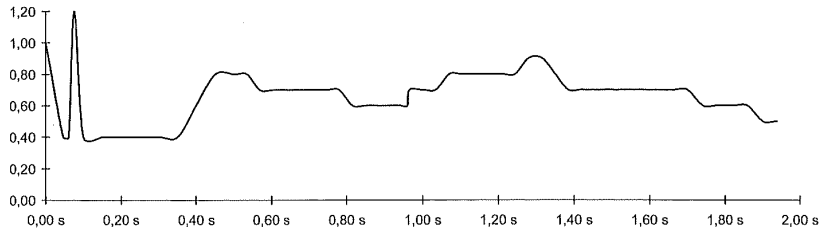


Bild 8.12 Andruck des Schreibstiftes im Verlauf über die Zeit beim Schreiben des Wortes „Rankl“.

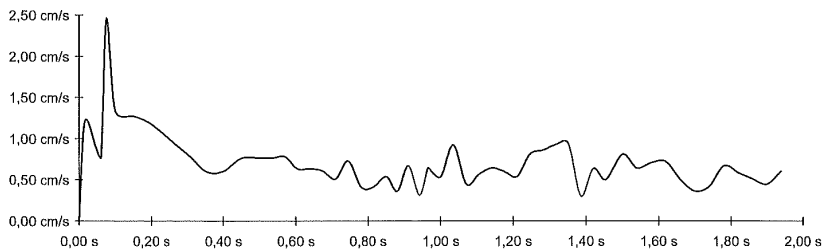


Bild 8.13 Geschwindigkeit des Schreibstiftes im Verlauf über die Zeit beim Schreiben des Wortes „Rankl“.

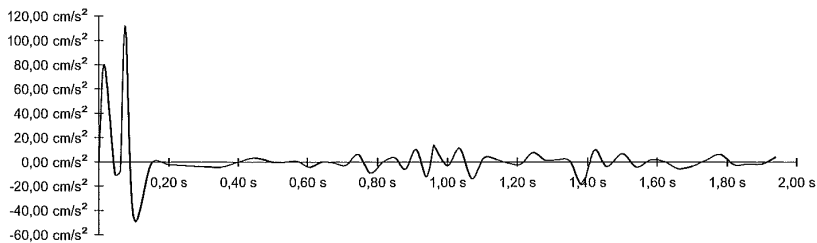


Bild 8.14 Beschleunigung des Schreibstiftes im Verlauf über die Zeit beim Schreiben des Wortes „Rankl“.

Die dynamische Unterschrift zur Identifizierung einer Person hat die höchste Akzeptanz aller Verfahren, da sie im täglichen Gebrauch in einer nahezu identischen Form von jedem verwendet wird. Aber auch hier sind die technischen Lösungen nicht einfach, da sich Unterschriften mit der Zeit ändern und nie völlig gleich sind. Man denke dabei nur an die Unterschiede der Schrift, die durch Schreiben im Stehen oder im Sitzen zustande kommen.

Tabelle 8.4 Vergleich der biometrischen Verfahren, die auf verhaltensgestützten Eigenschaften basieren. Die Werte sind typische Werte und können von Hersteller zu Hersteller stark differieren.

Methode	Dauer der Prüfung	Größe des Referenzwerts	Wahrscheinlichkeit für eine fehlerhafte Rückweisung (FRR)	Wahrscheinlichkeit für eine fehlerhafte Akzeptanz (FAR)
Stimme	5 s	100 Byte – 1 kByte	1 %	1 %
Dynamische Unterschrift	2 – 4 s	40 Byte – 1 kByte	1 %	0,5 %

8.2 Sicherheit einer Chipkarte

Die wesentliche Eigenschaft einer Chipkarte ist, dass sie für Daten und Programme eine sichere Umgebung bietet. Wäre es ohne größeren Aufwand möglich, Daten unbefugt aus einer Chipkarte auszulesen, würde sie sich nur mehr unwesentlich von einer Diskette unterscheiden, sie hätte lediglich eine andere Schnittstelle.

Natürlich ist es praktisch unmöglich, ein ganzes System oder auch nur eine Chipkarte so zu bauen, dass perfekte Sicherheit herrscht, die durch nichts und niemanden gebrochen werden kann. Schließlich muss man nur den Aufwand für den Angriff genügend hoch treiben, und dann kann man in jedes System eindringen oder es manipulieren. Doch jeder potenzielle Angreifer wird, bewusst oder unbewusst, für sich und seine Ziele immer eine Art Wertanalyse machen. Denn das Ergebnis, dass er durch das Brechen eines Systems erhält, muss die Arbeit, Geld und Zeit wert sein, die er dafür aufwendet. Falls das Resultat – sei es in monetärer Form oder auch reputationsmäßig in der (Fach-)Welt – den Aufwand nicht wert ist, wird niemand allzu viel Energie in das Brechen eines Systems oder einer Chipkarte stecken.

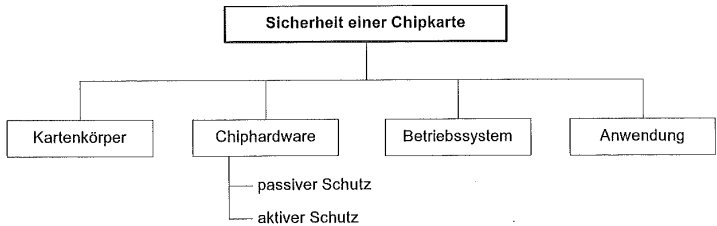


Bild 8.15 Klassifizierungsbaum der Sicherheit einer Chipkarte.

Die folgenden vier Segmente garantieren die Sicherheit einer Chipkarte: Das erste Segment der Sicherheit ist der Kartenkörper, in dem der Mikrocontroller eingebettet ist. Viele der dafür verwendeten Sicherheitsmerkmale sind nicht nur maschinenlesbar, sondern können zusätzlich auch noch visuell von Menschen geprüft werden. Die dabei angewandten Techniken sind nicht chipkartenspezifisch, sondern werden auch bei anderen Kartentypen eingesetzt. Die weiteren Segmente – Chiphardware, Betriebssystem und Anwendung – schützen die Daten und Programme im Mikrocontroller der Chipkarte.

Nur wenn all diese Segmente vorhanden sind und ihre Abwehrmechanismen funktionieren, ist die Sicherheit einer Chipkarte gewährleistet. Nutzt man die Chipkarte ausschließlich in einem Bereich ohne menschliche Kartenprüfung, so ist das Segment für den Kartenkörper nicht notwendig. Die drei vom Kartenkörper unabhängigen Segmente sind jedoch unabdingbar für die physikalische und logische Angriffssicherheit einer Chipkarte. Fällt auch nur eines dieser drei Segmente aus oder erfüllt die an sie gestellten Ansprüche nicht, ist die Chipkarte nicht mehr sicher, da diese drei Segmente mit einer logischen UND-Verknüpfung verbunden sind.

Die Lebensdauer typischer Chipkarten beträgt in der Regel 3 Jahre. Die Herausforderung für die Hersteller von Chipkarten-Mikrocontrollern und Chipkarten-Betriebssystemen besteht darin, um mindestens diese Zeitspanne allen Angreifern voraus zu sein. Damit können etwaige Angriffe bereits im Vorfeld mit geeigneten Gegenmaßnahmen in ihren Auswirkungen minimiert bzw. verhindert werden. Dieser zeitlicher Vorsprung ist jedoch nicht immer möglich weshalb bei Anwendungsarchitekturen immer streng darauf zu achten ist, dass eine einzelne erfolgreich angegriffene Chipkarte nicht das ganze jeweilige System diskreditiert.

8.2.1 Systematik der Angriffe und Angreifer

Das große Problem aller IT-Systeme, die Angriffen ausgesetzt sind, ist der oft auftretende Lawineneffekt bei geglückten Attacken. Schafft es beispielsweise ein Drucker, Banknoten in ausreichender Qualität zu fälschen und in Stückzahlen herzustellen, dann ist dies zwar für die jeweilige Nationalbank bedauerlich, doch führt es in der Praxis nie zu einer Inflation der Landeswährung. Der Fälscher wäre erstens gar nicht in der Lage, eine genügende Anzahl von Banknoten herzustellen, und zweitens ist es sehr risikoreich, die Fälschungen in großer Stückzahl in Umlauf zu bringen.

Bei elektronischem Geld verhält es sich jedoch etwas anders. Da dieses aus nichts anderem als aus immaterieller Information besteht, ist es in der Praxis unmöglich, Original und Kopie voneinander zu unterscheiden. Zudem kann es beim Bekanntwerden einer neu entdeckten Fälschungsmöglichkeit zu einem Lawineneffekt an Nachahmungstätern kommen. Die dabei auftretenden Effekte kann man in aller Deutlichkeit bei den in erheblichen Stückzahlen gefälschten vorbezahlten Telefonkarten studieren. Manche Netzbetreiber können sich nur noch durch an den Kartentelefonen eingeschränkte Gesprächszielorte gegen die Angriffe wehren.

Falls ein Designfehler oder eine Schwäche in einem großen Chipkartensystem bekannt wird, kann man davon ausgehen, dass diese Information innerhalb von Tagen oder wenigen Wochen weltweit über Internet gestreut wird. Sehr schnell wird auch entsprechende Software und – wenn nötig – auch Hardware angeboten, die in der Regel vollständig dokumentiert sind, sodass ein Nachbau problemlos möglich ist. Die Software wird üblicherweise auch im Quelltext veröffentlicht, sodass sie von Dritten weiterentwickelt werden kann, was in der Regel auch passiert. Damit setzt eine sehr rasche Evolution von Hard- und Software ein, die innerhalb kurzer Zeit auf den jeweiligen Zweck hochoptimiert wird.

Im Folgenden wird versucht, mögliche Angriffe und Angreifer auf Chipkarten zu systematisieren. Der Schwerpunkt ist natürlich auf die informationstechnische Seite von Chipkarten gelegt und nicht auf durch Menschen prüfbare Sicherheitsmerkmale des Kartenkörpers. Damit werden potenzielle Angriffe einschätzbar, und man kann dagegen Vorkehrun-

gen treffen. Eine Abwehr gegen unbekannte Gefahren ist bekanntlich weit schwieriger als gegen bekannte.

Bei der Einteilung der Angriffe haben wir uns an die ISO 13491-1 angelehnt, die Konzepte, Anforderungen und Evaluierungsmethoden von kryptografisch sicheren Geräten im Bankensektor beschreibt.

Einstufung der Angriffe

Es gibt unterschiedliche Ansätze zur Systematisierung von Angriffen auf Chipkarten. Bei Sicherheitsevaluierungen werden beispielsweise alle möglichen Angriffsarten nach der jeweiligen Phase des Kartenlebenszyklus gruppiert und formal beschrieben [IC Protection 97, Isselhorst 97]. Das Ergebnis besteht dann aus seitenlangen Listen, in denen für die jeweilige Phase alle denkbaren Angriffe gegenübergestellt sind. Bei der eigentlichen Evaluierung wird dann für jeden einzelnen Punkt geprüft, ob er vom System oder der Chipkarte abgewehrt werden kann. Analog der Fehlerbaumanalyse (*fault tree analysis*) kann man auch eine Angriffsbaumanalyse (*attack tree analysis*) aufbauen [Schneier 99]. Diese ist für Detailuntersuchungen und die Darstellung von Abhängigkeiten von großem Nutzen. In diesem Buch wurde eine andere Systematik gewählt, um das Thema möglichst realitätsnah darzustellen und auch anschaulich das Ping-Pong-Spiel von Angriff und Abwehr zu zeigen. Zudem soll hier ein allgemein gültiger und nicht systemspezifischer Überblick über Angriffs- und Abwehrmethodiken gegeben werden.

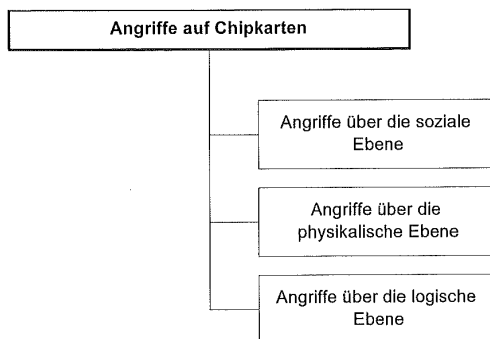


Bild 8.16 Klassifizierungsbaum zur Einstufung von Angriffen auf Chipkarten.

Prinzipiell lassen sich die Angriffe auf Chipkarten in drei verschiedene Arten einteilen: Angriffe über die soziale Ebene, die physikalische Ebene und die logische Ebene. Selbstverständlich gibt es in der Praxis auch Mischformen. So kann ein Angriff auf der physikalischen Ebene etwa dazu führen, dass damit ein nachfolgender Angriff auf der logischen Ebene vorbereitet wird, wie dies etwa bei der differenziellen Fehleranalyse der Fall ist.

Angriffe über die soziale Ebene sind Attacken, welche sich primär an die Menschen richten, die mit einer Chipkarte umgehen. Dies kann ein Entwickler beim Halbleiterhersteller sein, ein Softwaredesigner oder auch, später im Lebenszyklus, der Kartenbesitzer. Diese Angriffe können nur teilweise durch Technik abgefangen werden, hauptsächlich müssen hier organisatorische Maßnahmen abwehrend wirken. Das Ausspähen der PIN beim Eintip-

pen an einer Tastatur lässt sich sehr gut durch beiderseitige Sichtblenden am PIN-Pad verhindern. Angriffe über die soziale Ebene bei Programmierern von Chipkarten werden zwecklos gemacht, indem die benutzten Verfahren öffentlich sind und der erzeugte Programmcode noch zusätzlich von Dritten evaluiert wird. Die Sicherheit hängt damit nur an den geheimen Schlüsseln, und das Wissen des Entwicklers ist für einen Angreifer nutzlos.

Für Angriffe auf Chipkarten über die physikalische Ebene benötigt man in der Regel einiges an technischem Gerät, da man in irgendeiner Form an den Chipkarten-Mikrocontroller Hardware-technisch herankommen muss. Die Angriffe können entweder statisch am abgeschalteten Mikrocontroller oder dynamisch während des Betriebs stattfinden. Die statischen physikalischen Angriffe sind für den Attackierer nicht zeitkritisch, er kann in Ruhe seiner „Arbeit“ nachgehen. Bei den dynamischen Angriffen müssen jedoch ausreichend schnelle Messwert-Erfassungsgeräte zur Verfügung stehen.

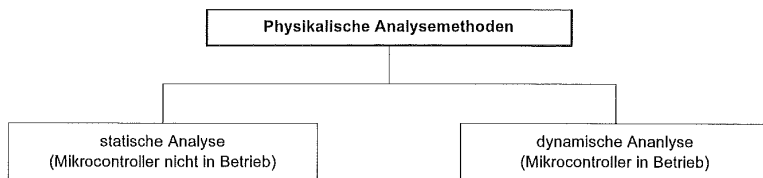


Bild 8.17 Klassifizierungsbaum der physikalischen Methoden zur Analyse von Chipkarten-Mikrocontrollern.

Die bisher bekannt gewordenen erfolgreichen Angriffe auf Chipkarten erfolgten meistens über die logische Ebene. Diese Angriffe kommen durch bloßes Nachdenken oder durch Berechnungen zustande. Die klassische Kryptoanalyse gehört hier ebenso dazu wie Angriffe über erkannte Fehler im Chipkarten-Betriebssystem oder trojanische Pferde im ausführbaren Code von Chipkarten-Anwendungen.

Analog zur Kryptoanalyse von kryptografischen Protokollen kann man Angriffe noch in passive und aktive Angriffe aufgliedern: Bei passiven Angriffen analysiert der Angreifer ohne direkte Veränderungen das Übertragungs- oder kryptografische Protokoll und nimmt beispielsweise Messungen auf dem Halbleiter vor. Im Gegensatz dazu wird bei einem aktiven Angriff manipulierend auf die Datenübertragung oder auf den Mikrocontroller zugegriffen.

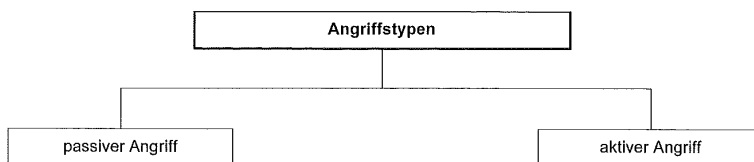


Bild 8.18 Klassifizierungsbaum der Typen von Angriffen auf Chipkarten.

Beim Zeitpunkt der möglichen Angriffe könnte man die Phasen des Kartenlebenszyklus nach ISO 10 202-1 verwenden.⁵ Dies wäre aber weitschweifig und langatmig, weshalb hier

⁵ Siehe auch Kapitel 10 „Lebenszyklus einer Chipkarte“.

zugunsten der Leserlichkeit eine kleine Vereinfachung vorgenommen wurde. Die Angriffe sind eingeteilt in die Zeiträume der Entwicklung, Produktion und Kartenbenutzung.

Angriffe bei der Entwicklung schließen sowohl Systemdesign, Chipentwicklung, Betriebssystementwicklung als auch Anwendungserstellung mit ein. Unter Produktion wird hier allgemein jeder Prozess verstanden, bei dem Hardware gefertigt wird. Damit ist der gesamte Bereich von der Waferfabrikation beim Halbleiterhersteller bis zur Kartenpersonalisierung und dem Kartenversand abgedeckt. Die Kartenbenutzung ist das Stadium, bei dem sich die Chipkarte im Feld befindet, sie also vom Benutzer verwendet wird.

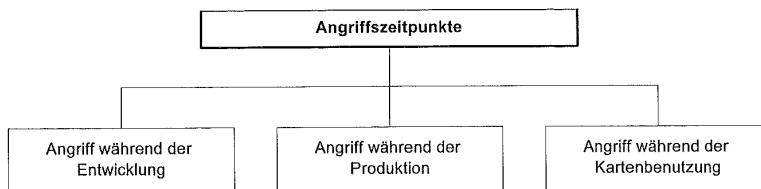


Bild 8.19 Klassifizierungsbaum der möglichen Angriffszeitpunkte.

Auswirkungen eines Angriffs und Einstufung der Angreifer

Um die Stärken und Schwächen von Angreifern auf die Sicherheit einer Chipkarte realistisch einschätzen zu können, ist es wichtig, sich zumindest ein grobes Bild der möglichen Angreifer zu machen. Diese Informationen kann man dann auch für die Konzeption von Abwehrstrategien und -mechanismen benutzen.

Üblicherweise haben typische Angreifer eine von zwei grundlegenden Motivationen für ihr Tun. Dies kann einerseits schlicht und einfach Geldgier sein oder andererseits das Bedürfnis, Ruhm und Reputation in der Szene zu erlangen. Die beiden Beweggründe haben unterschiedliche Konsequenzen für den Systembetreiber. Ein Angreifer, der mit seiner Tätigkeit Geld verdienen will, kann mit einem gewissen Risiko selber zum „Kartenherausgeber“ werden⁶ oder auch versuchen, den Systembetreiber zu erpressen. Beiden Szenarien kann durch die üblichen rechtlichen Mittel begegnet werden. Werden Einzelheiten des Angriffs öffentlich, ist der Ruf des Chipkarten-Systems geschädigt. Der schlimmste Fall der Rufschädigung des Systembetreibers tritt dann ein, wenn eine größere Anzahl von Kartenbenutzern Geld bei einem Angriff verliert.

Eine ähnliche Beeinträchtigung des Rufs eines Chipkarten-Systems entsteht auch, wenn der Angriff nicht aus kriminellen Neigungen, sondern aus wissenschaftlichem Forschungsdrang zustande kommt. Für einen Angreifer aus diesem Metier ist die Tätigkeit nur dann ein Erfolg, wenn er anschließend entsprechende Veröffentlichungen vorweisen kann. Dabei steht er auch noch unter enormem Zeitdruck, seine Entdeckung so schnell wie möglich zu publizieren, da in diesem Gewerbe bekanntermaßen immer nur der Erste den Ruhm erntet. Das Ergebnis besteht darin, dass sich der Systembetreiber unvermutet oder nach kurzer Vorwarnzeit mit einer Veröffentlichung konfrontiert sieht, in der im Detail ein Angriff auf sein System beschrieben steht. Anschließend wird der publik gewordene Angriff von Interessierten Schritt für Schritt weiter verfeinert und auch für Außenseiter verständ-

⁶ Dies ist der übliche Weg bei sich selbst wiederaufladenden Telefonkarten.

lich gemacht. Die Spitze besteht dann darin, dass Programme via Internet veröffentlicht werden, die den Angriff vollautomatisch durchführen.

Mit einem ähnlichen Ablauf sahen sich im Frühjahr 1998 einige GSM-Netzbetreiber konfrontiert. Der Angriff auf den Kryptoalgorithmus COMP 128, der für A3/A8 benutzt wurde hatte jedoch in diesem Fall keine großen nachteiligen Auswirkungen auf den regulären Netzbetrieb. Diese Vorgehensweise beinhaltet für den Angreifer einen wichtigen Aspekt. Er gilt als erfolgreicher Entdecker eines Sicherheitslecks, gehört somit zu den „Guten“ und muss fast nie rechtliche Konsequenzen seines Tuns fürchten.

Als Quintessenz kann aus diesen Szenarien abgeleitet werden, dass es für einen Systembetreiber letztendlich keine sonderlich große Rolle spielt, ob nun ein „Böser“ oder ein „Guter“ angreift. Der Schaden ist bei den wirklich gefährlichen Angriffen sowohl in finanzieller Hinsicht als auch in Hinblick auf den Ruf meist ziemlich groß. Im schlimmsten Fall muss das System außer Betrieb gehen und alle Karten müssen gesperrt und neue, gegen den Angriff immune ausgegeben werden. Bei großen System mit mehreren Millionen Chipkarten kann eine derartige Neuemission länger als ein halbes Jahr dauern.

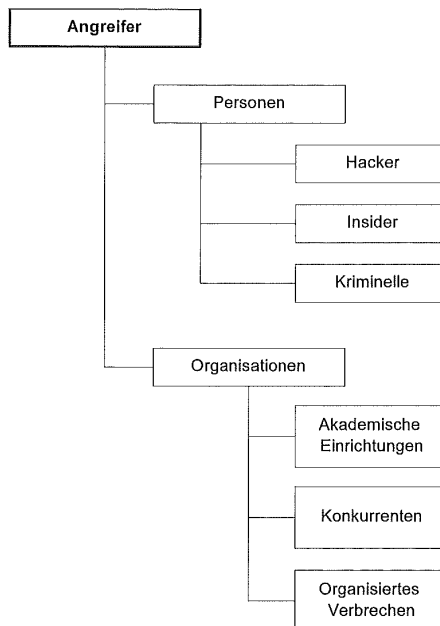


Bild 8.20 Klassifizierungsbaum der möglichen Typen von Angreifern.

Der Klassifizierungsbaum in Bild 8.20 zeigt unter Zugrundelegung der obigen Aspekte und Erfahrungen die möglichen Typen von Angreifern. Alle vier können gleichermaßen gefährlich für ein Chipkarten-System sein, doch haben sie unterschiedliche Fähigkeiten und Möglichkeiten. Der typische Hacker verfügt über ein mittleres Know-how hinsichtlich des Systems, gute kreative Ideen und in der Regel auch einen dazu passenden Freundeskreis. Ihm steht normalerweise kein gut ausgestattetes Labor zur Verfügung und auch seine

finanziellen Mittel sind beschränkt. Ist er jedoch qualifiziert, kann er durchaus bei geschicktem Vorgehen Zugriff auf große Rechenleistung, beispielsweise durch eine Aktion im Internet, bekommen.

Angriffe einer besonderen Kategorie sind alle Insider, bei denen vorausgesetzt werden muss, dass sie über sehr gute Systemkenntnisse verfügen. Unter Umständen haben sie Zugang zu Hard- und Softwarekomponenten und kennen vielleicht sogar Schwachstellen. Sofern es sich um Einzelpersonen handelt, kann man sie ansonsten in den ihnen zur Verfügung stehenden Mitteln und Möglichkeiten einem Hacker gleichsetzen. Insider sind allerdings weder anonym noch sehr zahlreich, weshalb Angriffe von ihnen gewöhnlich rückverfolgbar sind.

Die dritte Gruppe von Einzelpersonen, die als potenzielle Angreifer infrage kommen, sind Kriminelle. Diese besitzen in der Regel keine große Sachkenntnis, doch erhebliche Energie, um mit ihrem Tun Vorteile in meist finanzieller Hinsicht zu erlangen.

Ein in der Praxis nicht zu vernachlässigendes Angriffspotenzial stellen akademische Einrichtungen wie Universitäten, Hochschulinstitute mit ihren Studenten und Wissenschaftlern dar. Sie besitzen nicht unbedingt Spezialwissen über einen bestimmten Chipkarten-Mikrocontroller oder eine Chipkarten-Anwendung, doch großes allgemein anwendbares Wissen. Zudem stehen ihnen qualifizierte und billige Arbeitskräfte in Form von Studenten, Diplomanden und Doktoranden zur Verfügung sowie ausreichend technisches Equipment in den Labors. In vielen dieser Einrichtungen gibt es auch genügend Rechenleistung sowie hochmotivierte Menschen mit einem Hang zu Experimenten und Versuchen.

Eine besondere Art von Angreifern stellen Konkurrenten dar. Diese verfügen in der Regel über ein erhebliches Fachwissen und zum Teil hochwertige Infrastruktur für Analysen.

Ein ganz anderes Niveau für Angriffe auf Chipkarten-Systeme hat natürlich das organisierte Verbrechen. Das dort vorhandene Geld macht es möglich, entweder durch Kauf oder kriminelle Methoden alles an Wissen und Werkzeug zu beschaffen, was für einen erfolgreichen Angriff notwendig ist.

Einstufung der Angriffsattraktivität

Damit bereits im Vorfeld effektive Abwehrmaßnahmen gegen Angriffe möglich sind, sollte man für alle infrage kommenden Schwachstellen die Angriffsattraktivität ermitteln. Dies könnte man mathematisch objektiv mittels einer Wertanalyse durchführen, um so eine Rangfolge der wahrscheinlichen Angriffsziele zu errechnen. Im Folgenden wird ein vereinfachtes Schema dargestellt, das aber durchaus eine relativ gute Abschätzung der Attraktivität von Angriffen und damit der wahrscheinlichen Angriffsrichtungen ermöglicht. Naturgemäß wird ein Angreifer in der Regel so vorgehen, dass sein Aufwand für die Attacke möglichst gering ist. Die in Tabelle 8.5 aufgeführten sechs Kriterien werden bewusst oder unbewusst sein Handeln beeinflussen.

Ein Angriff wird für eine Person oder Organisation umso interessanter, je weniger spezifisches Wissen und Können dazu erforderlich ist. Ähnlich verhält es sich, wenn man zu einem erfolgreichen Angriff keine Geheimnisse in Erfahrung bringen muss. Dies steht im Übrigen nicht im Gegensatz zum Prinzip von Kerckhoff, welches aussagt, dass die Sicherheit nur von einem Schlüssel abhängen soll und nicht vom kryptografischen Algorithmus selbst. Kerckhoffs Prinzip bedeutet nämlich keineswegs, dass alles, was man weiß, veröffentlicht werden muss, damit ein System sicher ist. Viele Geheimnisse erschweren es ungemein, einen erfolgreichen Angriff zuwege zu bringen.

Gerade im Bereich der systematischen Schlüsselsuche spielt der notwendige Zeitbedarf eine bedeutsame Rolle. Das klassische Beispiel dazu ist ein Brute-force-Angriff auf einen Kryptoalgorithmus mit einer durchschnittlichen Dauer von 10000 Jahren, was für einen ernst zu nehmenden Angriff nicht ausreichen würde.

Die Angriffsattraktivität wird auch sehr stark von den notwendigen technischen Komponenten für den Angriff bestimmt. Es muss sich dabei jedoch nicht unbedingt um den Kauf eines Gerätes handeln, denn unter Umständen reicht es bereits aus, wenn man das Gerät mieten kann oder auch nur Zugang dazu erhält. Beispielsweise kostet eine Apparatur zur Erzeugung und exakten Steuerung von fokussierten Ionenstrahlen mehrere hunderttausend Euro, man kann sie aber auch tageweise an Forschungsinstituten mieten, und manche Studenten dürfen sie für ihre Forschungsarbeiten kostenlos nutzen.

Auch die Verfügbarkeit der anzugreifenden Komponente beeinflusst im starkem Maße die Anziehungskraft eines bestimmten Angriffs. Um beispielsweise ein kartengestütztes elektronisches Geldbörsensystem zu attackieren, kann man entweder bei sich zuhause seine persönliche Chipkarte mit ihren kartenindividuellen Schlüsseln analysieren oder das Gleiche mit einem Sicherheitsmodul mit den systemweiten Hauptschlüsseln beim Systembetreiber versuchen. Das Problem dabei ist jedoch die Tatsache, dass der Zugang zu diesem Sicherheitsmodul vielfach geschützt ist.

Dies ist übrigens einer der Gründe, warum Chipkarten für Pay-TV starken Angriffen ausgesetzt sind. Der Angreifer kann im heimischen Wohnzimmer in aller Ruhe die Kommunikation und das Verhalten seiner Chipkarte studieren und versuchen, dies mit einem Computer oder einer Elektronikbastelei nachzuahmen. Niemand beobachtet und stört ihn dabei. Würde man Ähnliches an einem Chipkartenterminal in einem Supermarkt versuchen, unterbände bereits die Kassiererin alle weiteren Experimente. Einen guten Überblick zum Thema Sicherheit von elektronischem Geld mit und ohne Chipkarten gibt [BIS 96].

Ein abschließendes Kriterium und von entscheidender Bedeutung ist natürlich der Wert des Ergebnisses der Angriffsbemühungen. Die Anstrengungen müssen sich entweder in finanzieller Form lohnen oder als Reputation in der Öffentlichkeit oder in Szenekreisen. Daraus lässt sich ableiten, dass Angriffe auf die diversen Feldversuche mit elektronischen Geldbörsen nur von Hackern und akademischen Kreisen drohen, da es viel zu wenige und meist auch zu einseitige Akzeptanzstellen (Bäcker, Kioske, u. Ä.) gibt, um durch einen ernsthaften Angriff zu viel Geld zu gelangen.

Tabelle 8.5 Schema zur Ermittlung des Aufwands für Angriffe auf Hard- oder Software von Sicherheitskomponenten anhand der dafür notwendigen Voraussetzungen.

Angriffsattraktivität	niedrig	mittel	hoch
notwendiges Wissen und Können	groß	mittel	klein
notwendige Geheimnisse	viele	mittel	wenige
notwendiger Zeitbedarf	groß	mittel	gering
Beschaffung der notwendigen technischen Geräte (Kauf oder Zugang)	schwierig	mittel	einfach
Zugang auf anzugreifende Komponente	schwierig	mittel	einfach
Wert des Ergebnisses (Geld, Ruf)	gering	mittel	groß

8.2.2 Angriffe und Abwehrmaßnahmen während der Entwicklung

Schon in der Phase der Entwicklung von Mikrocontroller-Hardware und der Software für das Chipkarten-Betriebssystem werden die unterschiedlichsten Sicherheitsmaßnahmen ergriffen. Sicherheit muss, analog der Qualität, von Anfang an bei einer Entwicklung berücksichtigt werden und lässt sich nicht nachträglich in ein Produkt hineindesignen.

Generell ist zu Angriffen in der Phase der Entwicklung anzumerken, dass der Zugang zu den entsprechenden Einrichtungen sehr schwierig und das erforderliche Know-how sehr groß ist, weshalb sich die Angriffsattraktivität auch entsprechend reduziert. Trotzdem ist das Gefährdungspotenzial eines erfolgreichen Angriffs in dieser Phase beträchtlich, da die Manipulationsmöglichkeiten sehr umfangreich sind.

8.2.2.1 Entwicklung des Chipkarten-Mikrocontrollers

Die hardwaretechnische Entwicklung eines Chipkarten-Mikrocontrollers dauert viele Monate und wird von wenigen Personen bei einem Halbleiterhersteller in zutrittsgesicherten und überwachten Räumen durchgeführt. Die entsprechenden Computersysteme für das Halbleiterdesign sind in der Regel in ein autarkes und vom Rest der Welt abgekoppeltes Netzwerk eingebunden. Damit wird verhindert, dass von außen Veränderungen am Chipdesign vorgenommen oder der interne Aufbau des Chips ermittelt werden kann.

Um Manipulationen, die sich dann schwächend auf die Sicherheit auswirken, an einem Chipdesign vornehmen zu können, bedarf es sehr umfangreichen Insiderwissens, weshalb diese Art von Angriff damit ziemlich unwahrscheinlich sein dürfte. Zudem werden mittlerweile beinahe alle Chipkartenhalbleiter von unabhängigen Prüfinstituten in Aufbau und Schutzmaßnahmen evaluiert, sodass ein Insiderangriff bekannt würde. Allerdings könnte es beispielsweise für einen Angreifer durchaus von Vorteil sein, wenn er die exakten Designkriterien und die Anordnung der Baugruppen auf dem Chip kennt, denn damit verfügt er beispielsweise auch über das Wissen hinsichtlich der vorhandenen Schutzelemente, Sensoren und des Scramblings von Bussen und Speichern. Diese Kenntnisse könnte ein Angreifer bei einer physikalischen Chipanalyse zu einem späteren Zeitpunkt nutzbringend einsetzen.

Schutz: Designkriterien

Für die Gestaltung der Funktionen eines Chipkarten-Mikrocontrollers gibt es einige grundlegenden Kriterien. Zum einen müssen die Maßnahmen gegen statische und dynamische Angriffe auch wirksam sein. Sensoren und Schutzelemente nützen wenig, wenn sie allzu leicht umgehbar sind oder unter Umständen nie wirksam werden. Als Beispiel seien hier Sensoren auf dem Halbleiterchip aufgeführt, die einen so großen Flächenbedarf haben, dass sie ohne Probleme mit einer Nadel zerstört werden können und somit ihrer Schutzfunktion nicht mehr nachkommen.

Ein von den üblichen Standardbausteinen unterschiedliches, doch sehr wichtiges Designkriterium ist die Forderung, dass keinesfalls undokumentierte Mechanismen oder Funktionen („*that's not a bug, that's a feature*“) vorhanden sein dürfen. Solche undoku-

mentierten „Features“ werden meistens nie vollständig getestet, da sie nur wenige Personen kennen, und weisen insofern oft noch diverse Fehler und Schwachstellen auf. Da sie nicht dokumentiert sind, könnten sie versehentlich bei der Evaluierung der Hardware übersehen und später unter Umständen für Angriffe genutzt werden. Deshalb ist die Verwendung solcher undokumentierten Features streng untersagt, auch wenn sie für die Entwickler oftmals hilfreich sein könnten.

Schutz: Eindeutige Chipnummer

Bei der Entwicklung des Halbleiters müssen alle hardwaretechnischen Sicherheitselemente konzipiert und für den späteren Mikrocontroller umgesetzt werden. Neben den Sensoren und Schutzschichten ist ein Element dabei ein WORM-Speicher (*write once, read multiple*), oft auch OTP-Speicher (*one time programmable*) genannt. In diesen wird bei der Halbleiterproduktion eine eindeutige Chipnummer geschrieben. Damit ist der Chip individualisiert, auch eindeutig weiterverfolgbar, und die spätere Chipkarte kann dann systemweit eindeutig identifiziert werden. Zudem lässt sich diese Nummer auch zur Ableitung von Schlüsseln verwenden und bietet die Möglichkeit, Sperrlisten anzulegen, mit denen verdächtige Chipkarten aus dem Verkehr gezogen werden können.

Man darf dabei jedoch nicht übersehen, dass sich diese Nummer zwar nicht mehr auf einem Originalchip ändern lässt, doch schützt sie natürlich nicht vor einer Nachahmung des Chips durch einen frei programmierbaren Mikrocontroller. Deshalb dürfen Sicherheitsmechanismen nicht darauf beruhen, dass ein bestimmter Chip eine bestimmte Nummer in seinem WORM-Speicher hat. Diese eindeutige Nummer kann nur eine Grundlage für echte kryptografische Sicherheitsmechanismen sein. So könnte die Chipnummer für die Ableitung von geheimen Schlüsseln benutzt werden, die dann wiederum in einer Challenge-Response-Authentisierung benutzt werden.

8.2.2.2 Entwicklung des Chipkarten-Betriebssystems

Die Entwicklung von Software für Chipkarten verläuft analog den modernen Entwicklungsprinzipien von Software. Unabhängig davon, welche Entwicklungsmethodiken (Wasserfallmodell, Spiralmodell, ...) verwendet werden, sind allerdings bestimmte Rahmenbedingungen einzuhalten.

Für die Entwicklungsrechner ist auf jeden Fall ein eigenes, vollständig abgeschottetes Netzwerk erforderlich, das keine Zugriffe von außen zulässt. Die Entwicklungswerkzeuge, wie Compiler und Chipsimulatoren, sind Software-Pakete, deren Funktionsfähigkeit in eigenen Tests geprüft wurde. Manchmal werden sogar zwei unterschiedliche Compiler benutzt, um sicherzustellen, dass das Ergebnis korrekt ist. Die Benutzung von Software, deren Herkunft nicht exakt nachweisbar ist, ist grundsätzlich untersagt, da dies ein möglicher Weg wäre, um ein Entwicklungswerkzeug zu manipulieren und dadurch in Folge den zu erstellenden Programmcode zu verändern.

Schutz: Entwicklungsprinzipien

Bei der Softwareentwicklung dürfen, ähnlich wie bei der Hardwareentwicklung, keine undokumentierten Features eingebaut werden. So wären beispielsweise, um die bei Chipkarten üblichen, aber aufwendigen Blackbox-Tests in Whitebox-Tests umzuwandeln,

manchmal durchaus Kommandos angebracht, mit denen beliebige Speicherbereiche ausgelesen werden können. Würde jedoch eines dieser Kommandos im Programm vergessen, ließen sich damit bei echten Chipkarten geheime Schlüssel auslesen. Um diesen Angriff schon im Ansatz zu unterbinden, ist die Erstellung von Dump-Kommandos unerwünscht, selbst wenn damit teure Entwicklungszeit gespart werden könnte. Termindruck und die stetig steigende Komplexität der Chipkarten-Betriebssysteme haben jedoch zur Aufweichung dieses Prinzips geführt. Um sicherzustellen, dass keinesfalls diese entwicklungsbegleitenden Kommandos in echten Chipkarten ins Feld kommen, werden in der Chipkarten-Fertigung spezielle Tests auf Nichtvorhandensein dieser Kommandos durchgeführt.

Ein weiteres Prinzip besteht darin, dass ein Programmierer niemals alleine an einer Programmentwicklung arbeitet. Dies verbietet sich eigentlich schon aus Sicht der Software-Qualitätssicherung, doch auch aus Gründen der Sicherheit gegenüber Angriffen muss hier immer das Vier-Augen-Prinzip gelten. Damit lassen sich Angriffe durch Insider wirkungsvoll erschweren, da sich immer mindestens zwei Entwickler in ihrem Tun einig sein müssen. Zusätzlich werden noch regelmäßig interne Source-Code-Inspektionen durchgeführt, die sowohl zur Sicherstellung der Qualität und Überwachung des Entwicklungsprozesses dienen.

Ist die Softwareentwicklung abgeschlossen, findet häufig eine Prüfung des gesamten erstellten Sourcecodes und auch der Funktionalität durch unabhängige Prüfinstitute im Rahmen einer Software-Evaluierung statt.⁷ Die Gründe für diese zeit- und kostenintensiven Prüfungen liegen vor allem in der Kontrolle auf Softwarefehler, doch bewirken sie auch, dass es für einen Entwickler nicht mehr möglich ist, beispielsweise ein trojanisches Pferd im Betriebssystem zu verstecken. Diese lassen sich in der Praxis eigentlich nur mittels Durchsicht des kompletten Programmcodes finden, da ein erfahrener Programmierer durchaus in der Lage wäre, Mittel und Wege zu finden, um ein trojanisches Pferd so zu verstecken, dass es durch Blackbox-Tests nicht mehr gefunden werden kann.

Schutz: Aufteilung von Wissen

Arbeiten mehrere Personen an einer Aufgabe, ist das Ergebnis aufgrund der unterschiedlichen Erfahrungen und Meinungen wesentlich robuster gegenüber Angriffen. Das Prinzip der Wissensteilung (*shared secret*) wirkt diesem Ansatz des „jeder weiß alles über alles“ entgegen. Grundsätzlich sollte bei der Entwicklung von Sicherheitskomponenten nicht das gesamte Wissen auf eine einzelne Person konzentriert sein, da sie damit auch angreifbar wird. So wird, ähnlich wie in manchen militärischen Bereichen, das Wissen bei der Entwicklung auf mehrere Gruppen von Personen verteilt, sodass zwar Diskussionen zwischen Experten über ein Thema möglich sind, doch es niemanden gibt, der alles weiß.

Analog Obigem verhält es sich mit der Komplettierung des Chipkarten-Betriebssystems, d. h. dem Laden von Tabellen, Programmcode und Konfigurationsdaten ins EEPROM. Dies stellt neben der erhöhten Flexibilität auch einen Sicherheitsaspekt dar. Das gesamte Wissen über das Betriebssystem befindet sich dann beim Chiphersteller, der den kompletten und assemblierten ROM-Programmcode zur Maskenherstellung erhält. Die Teile des Betriebssystems, die sich im EEPROM befinden, sind dem Chiphersteller nicht bekannt, sodass er bei einer Analyse des ROM-Codes nicht die gesamten Sicherheitsmechanismen und die Funktionalität des Betriebssystems erfahren kann.

⁷ Siehe auch Abschnitt 9.3 „Evaluierung und Test von Software“.

8.2.3 Angriffe und Abwehrmaßnahmen während der Produktion

Angriffe während der Produktion der Chips bzw. der Chipkarten sind typische Insider-Angriffe, da die betreffenden Produktionsumgebungen geschlossene Umgebungen sind. Der Zugang ist stark reglementiert, und jeder Zutritt wird aufgezeichnet. Trotzdem darf man die Produktion nicht bei den Sicherheitsbetrachtungen aussparen, da hier auch einige technisch sehr interessante und wirkungsvolle Angriffe durchgeführt werden könnten.

Schutz: Authentisierung bei den Fertigungsschritten

Schon bei der Waferherstellung sind die Chipkarten-Mikrocontroller durch eine Chipnummer individualisiert und mit einem Transportcode geschützt. Bei den neueren Betriebssystemen ist der Transportcode chipindividuell, und bei jedem fertigungstechnischen Zugriff auf den Chip ist eine Authentisierung zwingend erforderlich. Dies macht zwar die Fertigung etwas aufwendiger und erfordert natürlich ein Sicherheitsmodul an den jeweiligen Maschinen, doch erhöht es in erheblichem Maße die Sicherheit.

Ein nahe liegender Angriff in der Fertigung ist die Einschleusung von Dummy-Chips oder Dummy-Chipkarten, die sich identisch den regulären Bauteilen verhalten, jedoch beispielsweise ein Kommando zum Speicherdump haben. Der Austausch eines echten Chip durch einen Dummy-Chip ist natürlich frühestens nach der Trennung der Wafer in einzelne Dice möglich. Anhand einer Chipkarte für digitale Signaturen⁸ soll dieser Angriff hier kurz verdeutlicht werden. Der Angreifer tauscht bei der Initialisierung eine echte Chipkarte gegen eine Dummy-Karte aus. Diese Karte wird dann mit echten Daten initialisiert und anschließend personalisiert.

Da diese Chipkarte alle Funktionen einer echten Chipkarte aufweist, würde dann auch die Generierung der Schlüssel für den asymmetrischen Kryptoalgorithmus im Mikrocontroller durchgeführt. Die dafür notwendigen Daten bekommt er über die Initialisierungs- und Personalisierungsdaten. Anschließend müsste es der Angreifer noch schaffen, seine Chipkarte wieder in Besitz zu bekommen, und er könnte den geheimen Signaturschlüssel aus der Karte mit seinem speziellen Dumpkommando auslesen. Da der dazugehörige öffentliche Schlüssel vom Trustcenter unterschrieben wurde und damit als echt bestätigt ist, ist der Angreifer nun im Besitz aller Informationen, um beliebig viele Duplikate einer als echt anerkannten Karte herzustellen.

Dieser Angriff ist schon deshalb unrealistisch, weil bereits in organisatorischer Hinsicht verhindert wird, Chips oder Chipkarten in die entsprechenden Fertigungsstätten hinein- oder herauszubringen. Zusätzlich verhindert die bei allen Fertigungsschritten notwendige Authentisierung zwischen Chipkarte und dem Sicherheitsmodul der Fertigungsmaschine einen Tausch der Chips oder Karten.⁹

⁸ Siehe auch Abschnitt 13.6 „Digitale Signatur nach deutschem Signaturgesetz“.

⁹ Eine ausführliche Darstellung der üblichen kryptografischen Verfahren bei Initialisierung und Personalisierung von Chipkarten findet sich unter Abschnitt 10.4 „Phase 3 des Lebenszyklus im Detail“.

8.2.4 Angriffe und Abwehrmaßnahmen während der Kartenbenutzung

Der Zugang zur anzugreifenden Komponente – die Chipkarte – erfordert für Angreifer nach Ausgabe der Chipkarten in aller Regel sehr viel weniger Aufwand als in den vorangehenden Phasen des Lebenszyklus. Dies ist einer der Gründe, warum gerade in der Phase der Kartenbenutzung die Wahrscheinlichkeit von Angriffen relativ groß ist.

Das Thema „Selbstzerstörung eines Chipkarten-Mikrocontrollers“ taucht in vielen Veröffentlichungen immer wieder als das ultimative Allheilmittel gegen alle Arten von Angreifern auf. Es gibt Hardwaresicherheitsmodule, z. B. für militärische Anwendungen, in denen solche Mechanismen manchmal eingesetzt werden, bei Chipkarten jedoch verbietet sich diese Abwehrmaßnahme aus mannigfaltigen Gründen. Sobald eine Chipkarte stromlos ist, hat sie erstens keine Möglichkeit, einen potenziellen Angriff zu erkennen, und zweitens besteht keine Chance für irgendwelche aktiven Abwehrmaßnahmen, da sie keinen Energiespeicher besitzt. Abgesehen davon, dass eine echte Selbstzerstörung aus rechtlichen Gründen wohl keinem Kartenbenutzer zugemutet werden kann. Wer würde denn für den Schaden aufkommen, der unter widrigen Umständen entstehen könnte, nur weil sich eine Chipkarte irtümlich selber zerstört hat? Zudem ist eine echte Selbstzerstörung überhaupt nicht notwendig, da es in beinahe allen Fällen reicht, die geheimen Schlüssel in der Karte zu löschen.

Es gibt aber zu diesem Thema noch einen weiteren Aspekt, und der betrifft das Löschen von Schlüsseln oder Sperren der Chipkarte. Es ist sehr schwierig für eine Chipkarte, überhaupt zu erkennen, ob sie angegriffen wird. Es gibt schlichtweg keinen Sensor, der anzeigt „Angriff – alles löschen“. Unterspannung oder zu hohe Taktfrequenzen könnten beispielsweise Indizien sein, doch sie treten auch im Normalbetrieb durch widrige Umweltbedingungen auf. Verschmutzte Kontakte führen zu einem hohen Übergangswiderstand und dies wiederum zu Unterspannung. Eine überhöhte Taktfrequenz kann an einem Chipkarten-Terminal auftreten, das für eine höhere Frequenz vorgesehen ist. Da es so schwierig und meistens sogar unmöglich ist, einen echten Angriff zu erkennen, verzichtet man in der Regel auf alle Mechanismen zur automatischen Blockierung der Karte oder zum Löschen der Schlüssel.

Im folgenden Abschnitt sind beispielhaft einige fast schon klassisch zu nennende Angriffe aufgeführt und erläutert. Die Angriffsbeschreibungen repräsentieren sozusagen den Stand der Technik und sollen vor allem in der Thematik der Chipkartensicherheit Unerfahrenen einen kompetenten Überblick geben, damit bereits bekannte kritische Mechanismen nicht aus Unkenntnis abermals verwendet werden. Um sich gegen diese Angriffe zu verteidigen, gibt es die beschriebenen Abwehrmaßnahmen. Diese können wiederum mit etwas veränderten Angriffsszenarien umgangen werden, woraus sich das bekannte Katz- und Mausspiel von Maßnahmen und Gegenmaßnahmen bei Angriff und Abwehr ergibt.

Die dargestellten Szenarien sind keine Anleitung zum Brechen der Sicherheit eines Chipkarten-Systems, da sie ausnahmslos bekannt und auch öffentlich sind [Kömmerring 99]. Für die Sicherheit heutiger moderner Chipkarten stellen sie keine ernst zu nehmende Bedrohung dar, da die beschriebenen Angriffe durch entsprechende Schutzmaßnahmen längst berücksichtigt wurden. Vor einigen Jahren hätte man damit aber vielleicht noch einige Erfolge erzielt.

Die Angriffe sind eingeteilt in solche, die auf die Chiphardware erfolgen, und in solche, bei denen auf logischer Ebene versucht wird, das Chipkarten-System zu brechen. Die physikalischen Angriffe oder Analysemethoden lassen sich noch in statische oder dynamische

unterteilen. Bei einer statischen Analyse ist der Chip nicht in Betrieb, er kann aber elektrisch angeschaltet sein. Bei der ungleich schwerer durchzuführenden dynamischen Analyse eines Chips arbeitet dieser während der Analyse in vollem Funktionsumfang.

Tabelle 8.6 Überblick von typischen Angriffen, die auf Systeme mit Chipkarten Einfluss hatten, geordnet nach Zeitpunkt des Bekanntwerdens. Im folgenden Text sind die genannten Angriffe und die dazugehörigen ersten Gegenmaßnahmen detaillierter beschrieben.

bekannt seit	Angriff	Kurzbeschreibung des Angriffs
vor 1990	Abhören der Datenübertragung	Durch an das Modul angebrachte Drähte ist es möglich, die Datenübertragung zwischen Terminal und Karte abzuhören. Die Gegenmaßnahme dazu war die Einführung von Secure Messaging.
≈ 1990	Auflösen der Passivierung	Das Auflösen der Passivierungsschicht über dem Mikrocontroller ist die Voraussetzung für physischen Zugang zu den Komponenten auf dem Mikrocontroller. Die Gegenmaßnahme dazu war die Einführung von Passivierungsdetektoren auf den Mikrocontrollern.
≈ 1990	Manipulation der Datenübertragung	Durch elektrische Isolierung der Kontaktfelder des Moduls und entsprechend am Modul angebrachte Drähte ist es möglich, die Datenübertragung zwischen Terminal und Karte beliebig zu manipulieren. Die Gegenmaßnahme dazu war die Einführung von Secure Messaging.
≈ 1991	Löschen des EEPROM durch UV-Licht	Durch Löschen des EEPROMs mittels UV-Licht können beispielsweise Zähler wieder auf ihren Ausgangswert zurückgesetzt werden. Die Gegenmaßnahme dazu war die Einführung von Lichtsensoren auf den Mikrocontrollern.
≈ 1991	Ersatzschaltung von Speicherkarten	Mittels Ersatzschaltungen von Speicherkarten kann sowohl die Funktionalität der Speicherkarte als auch das geheime Echtheitsmerkmal emuliert werden. Die Gegenmaßnahme dazu war die Einführung von Challenge-Response-Authentisierung auf den Speicherkarten.
≈ 1992	Abschalten der Spannungsversorgung	Durch Abschalten der Spannungsversorgung bei der PIN-Prüfung kann das Schreiben des Fehlbedienungszählers verhindert werden. Die Gegenmaßnahme dazu war, den Fehlbedienungszähler vor der PIN-Prüfung prophylaktisch zu erhöhen.
≈ 1993	Takt anhalten	Durch Anhalten der Taktfrequenz und Analyse des RAM mittels Elektronenstrahltester können Rückschlüsse auf den RAM-Inhalt gezogen werden. Die Gegenmaßnahme dazu war die Einführung von Unterfrequenzdetektoren auf den Mikrocontrollern.
≈ 1993	Manipulation des Mikrocontrollers durch Lasercutter	Die Komponenten auf dem Mikrocontroller können mit Lasercutter manipuliert werden. Die Gegenmaßnahme dazu war die Einführung von Schutzschichten über den Mikrocontrollern.
1995	Timing-Attack	Aufgrund von Unwissen wurde bei der Implementierung vieler Kryptoalgorithmen eine Abhängigkeit zwischen Schlüssel und Laufzeit geschaffen. Dies kann zur Ermittlung der geheimen Schlüssel genutzt werden. Die Gegenmaßnahme dazu war die Realisierung von rauschfreien Kryptoalgorithmen.

Tabelle 8.6 (Fortsetzung)

bekannt seit	Angriff	Kurzbeschreibung des Angriffs
≈ 1995	Abhören des Bus mit Mikroproben-nadeln	Die Busse auf dem Mikrocontroller können mit Mikroproben-nadeln abgehört werden. Die Gegenmaßnahme dazu war das Scrambling der Busse auf den Mikrocontrollern.
1996	DFA	Geheime Schlüssel von Kryptoalgorithmen können durch selektive Einstreuung von Fehlberechnungen des Prozessors berechnet werden. Die Gegenmaßnahmen dazu war die Einführung von Glitchdetektoren auf den Mikrocontrollern sowie entsprechende Vorsorgemaßnahmen in den Kryptoalgorithmen.
≈ 1996	Manipulation des Mikrocontrollers durch FIB	Der Komponenten auf dem Mikrocontroller können mit FIB manipuliert werden. Die Gegenmaßnahme dazu war die Einführung von Schutzschichten über dem Mikrocontroller.
1997	erschöpfende Schlüsselsuche beim DES	Durch leistungsfähige Rechner bzw. Netzwerke von Rechnern können DES-Schlüssel innerhalb einiger Stunden durch eine Brute-force-Attack errechnet werden. Die Gegenmaßnahme dazu war die Verwendung von Triple-DES.
1997	statistische Verteilung von PIN	Die Generierung der vierstelligen PINs im deutschen ec-Kartensystem wies keine statistische Gleichverteilung auf, weshalb manche PIN-Werte deutlich öfter vorkamen als andere. Die Gegenmaßnahme dazu war die Verwendung eines verbesserten Generierungsalgorithmus.
1998	SPA/DPA	Aufgrund der Stromaufnahme des Prozessors lassen sich die verarbeiteten Daten ermitteln. Die Gegenmaßnahmen dazu waren: Einführung von zufallsgesteuerten Wartezeiten des Prozessors, Prozessoren mit immer gleicher Stromaufnahme sowie eine große Zahl von Vorsorgemaßnahmen in der Software auf dem Mikrocontroller.
1998	COMP 128	Aufgrund einer Designschwäche des von einigen Netzbetreibern verwendeten Authentisierungsalgorithmus COMP 128 ist es möglich, durch eine Brute-force-Attack die geheimen Schlüssel zu ermitteln. Die Gegenmaßnahme dazu war die Verwendung von anderen Authentisierungsalgorithmen und die Begrenzung der Anzahl der Authentisierungen.
1998	Störung des Prozessors	Durch Störung des Prozessors (z. B. durch Lichtblitze) durch beispielsweise Lichtblitze kann dieser bei der Abarbeitung des Maschinenprogramms an kritischen Stellen gestört werden. Die Gegenmaßnahmen dazu waren entsprechende Detektoren auf den Mikrocontrollern sowie eine große Zahl von Vorsorgemaßnahmen in der Software.

8.2.4.1 Angriffe auf der physikalischen Ebene

Für Manipulationen im Bereich des Halbleiters ist ein großer technischer Aufwand nötig. Je nach Angriffsszenario sind dies Mikroskope, Lasercutter, Mikromanipulatoren, fokussierte Ionenstrahlen, Anlagen für chemische Abtragverfahren und sehr schnelle Computer für die Analyse, Protokollierung und Auswertung der elektrischen Vorgänge auf dem Chip. Diese Geräte und das Wissen um deren Anwendung stehen nur sehr wenigen Spezialisten oder Organisationen zur Verfügung, was die Wahrscheinlichkeit eines Angriffs auf physikalischer Ebene stark reduziert. Doch muss ein Karten- bzw. Halbleiterhersteller

grundsätzlich davon ausgehen, dass ein potenzieller Angreifer alle notwendigen Geräte und Vorrichtungen für einen solchen Angriff einsetzen kann und dementsprechende Sicherungen in die Hardware einbauen.

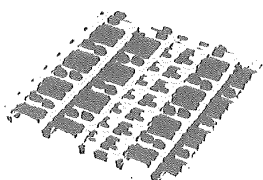


Bild 8.21 Grafische Darstellung des mit einem Atomic Force Mikroskop gemessenen Oberflächenprofils eines Chipkarten-Mikrocontrollers. Der dargestellte Höhenunterschied beträgt im Maximum lediglich 2,3 μm . (Giesecke & Devrient)

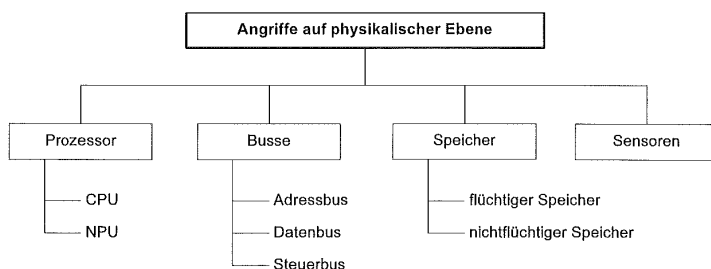


Bild 8.22 Klassifizierungsbaum der Ansatzpunkte für Angriffe auf physikalischer Ebene auf einem Chipkarten-Mikrocontroller.

Um Angriffe auf der physikalischen Ebene durchführen zu können, sind vorab einige Schritte notwendig: Als Erstes ist es notwendig, das Modul aus der Karte auszubauen, was mit einem scharfen Messer problemlos möglich ist. Danach muss der mit Epoxidharz auf dem Modul vergossene Chip freigelegt werden. Anderson und Kuhn [Anderson 96b] verwendeten dazu rauchende Salpetersäure, eine IR-Lampe als Wärmequelle und anschließend Azeton zum Reinigen des Chips. Anschließend ist der Halbleiterchip freigelegt und noch voll funktionsfähig. Manche Leute glauben, nun liegt der Chip schutzlos vor ihnen und muss nur noch „ausgelesen“ werden, doch ist dies keinesfalls so. Ein Angreifer muss sich nämlich nun durch die mannigfaltigen Sicherheitsmaßnahmen arbeiten, um Zugang zu den Geheimnissen zu bekommen.

Die Schutzelemente im Hardwarebereich können in passive und aktive Komponenten unterteilt werden. Passive Komponenten basieren direkt auf der Technologie der Halbleiterherstellung. Sie umfassen alle Möglichkeiten und Verfahren, die verwendet werden können, um die Speicherbereiche und die restlichen Funktionselemente des Mikrocontrollers vor den verschiedenen Arten der Analyse zu schützen.

Ergänzend zu den halbleitertechnischen (passiven) Möglichkeiten, bietet sich noch die ganze Palette an aktiven Elementen auf einem Silizium-Chip an. Aktiver Schutz bedeutet die Integration unterschiedlicher Sensoren auf dem Siliziumkristall. Diese Sensoren werden von der Software der Chipkarte bei Bedarf abgefragt und bewertet. Dies ist natürlich

nur dann möglich, wenn der Chip mit allen seinen Versorgungsleitungen voll aktiviert ist. Ein stromloser Chip kann keine Sensorsignale auswerten, geschweige denn bewerten. Manchmal ist gerade im Bereich der Sensorik der Sprung von notwendigen Schutzelementen bis zur technischen Spielerei oft nicht weit. Ein Lichtsensor auf einem Chip, der eine optische Analyse der Speicher verhindern soll, reagiert nicht, wenn sich der Chip ohne Spannung und Taktversorgung auf dem Objektträger eines Lichtmikroskops befindet. Auch würde sich solch ein optisch sehr gut auf dem Chip identifizierbarer Sensor sehr gut mit einem Tropfen schwarzer Tusche abdecken lassen, sodass seine Schutzfunktion auch im Betrieb ohne Aufwand neutralisiert werden kann. Dies lässt sich jedoch dadurch umgehen, dass eine Vielzahl von Lichtsensoren über den ganzen Chip verteilt sind.

Auch die langfristige Funktionssicherheit ist von Bedeutung. Beispielsweise trägt ein Temperatursensor, der aufgrund einer kurzzeitigen und nicht schädlichen Überhitzung über die Software der Chipkarte veranlasst, das gesamte EEPROM aus Sicherheitsgründen zu löschen, in keinem Fall zu einer erhöhten Funktions- oder Angriffssicherheit bei. Aus diesem Grund aktiviert man im Regelfall in den Chipkarten-Mikrocontrollern nur wenige Sensoren.

Nachstehend sind die wichtigsten und in der Praxis am häufigsten angewandten Schutzmechanismen von Chipkarten-Mikrocontrollern erläutert.

8.2.4.1.1 Statische Analysen am Chipkarten-Mikrocontroller

Schutz: Halbleitertechnologie

Die Strukturen auf dem Chip (Breite der Leiterbahnen, Größe der Transistoren etc.) bewegen sich an der Grenze des derzeit technisch Machbaren. Die üblichen Strukturbreiten liegen im Bereich zwischen $0,35\text{ }\mu\text{m}$ und $0,13\text{ }\mu\text{m}$, was an sich keine technische Besonderheit mehr darstellt. Die Transistordichte auf dem Silizium gehört jedoch zu den höchsten,

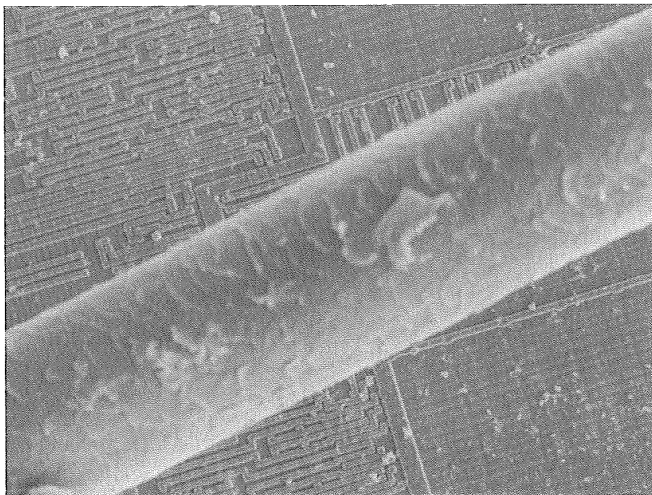


Bild 8.23 Menschliches Kopfhair im Vergleich zu Halbleiterstrukturen auf einem Chipkarten-Mikrocontroller in 1000-facher Vergrößerung. (Giesecke & Devrient)

die momentan mit den üblichen lithographischen Herstellungsverfahren möglich sind. Alleine diese sehr feinen Strukturen machen es beinahe unmöglich, mit Analyseverfahren Informationen aus dem Chip herauszuholen, weshalb Halbleitertechnologien mit Strukturgrößen von einem Mikrometer zurzeit noch als sicher angesehen werden. In Zukunft wird sich dies gewiss reduzieren.

Schutz: Chipdesign

Für das Design von halbleitertechnischen Bauelementen werden oft so genannte Standardzellen benutzt, die beispielsweise einen Prozessorkern oder bestimmte Speichertypen beinhalten. Der Vorteil besteht darin, dass ein Halbleiterhersteller mit diesen Standardelementen schnell und in hoher Qualität eine Vielfalt von unterschiedlichen Chips erstellen kann. Dieses für sicherheitsunkritische Massenartikel entwickelte Verfahren darf bei Chipkarten-Mikrocontrollern nicht eingesetzt werden, da Standardzellen in ihrem Aufbau und ihrer Funktionsweise bekannt sind und so einem potenziellen Angreifer zu viel Information in die Hände gegeben und somit die Arbeit wesentlich erleichtert würde. Die Komponenten für Chipkarten-Mikrocontroller sind speziell für diese entwickelt und werden auch nicht für andere Zwecke eingesetzt.

Schutz: Dummy-Strukturen

Ein in Expertenkreisen oft kontrovers diskutiertes Mittel gegen Angriffe sind Dummy-Strukturen auf dem Chip. Dies sind Elemente auf dem Halbleiter, die keine eigentliche Funktion haben, sondern einen Angreifer verwirren und ablenken sollen. Die Sicherheit beruht ganz und gar darauf, dass Existenz und Position auf dem Chip geheim gehalten werden. Dummy-Strukturen lassen sich zusätzlich noch überwachen, um auf diese Weise eine Veränderung zu detektieren und den Chip daraufhin abzuschalten. Der größte Nachteil dieser funktionslosen Strukturen ist ihr zusätzlicher Platzbedarf auf dem Halbleiter.

Schutz: Busse auf dem Chip

Alle internen Busse auf dem Chip, die den Prozessor mit den drei verschiedenen Speichertypen ROM, EEPROM und RAM verbinden, sind nicht nach außen geführt und dadurch nicht kontaktierbar. Es besteht also für einen Angreifer keine Möglichkeit, den Adress-, Daten- oder Steuerbus des Mikrocontrollers abzuhören oder zu beeinflussen und dadurch Speicherinhalte auszulesen. Überlicherweise werden die Busse in den unteren Schichten des Halbleiters aufgebaut, sodass es schwierig ist, sie direkt von der Oberfläche her zu kontaktieren. Zudem sind die Busse auf dem Chip statisch, chipindividuell oder sitzungsindividuell gescrambelt, sodass die Funktion der einzelnen Busleitungen von außen nicht erkennbar ist. Es gibt Chipkarten-Mikrocontroller bei denen das Scrambling der Busse sogar während der Sitzung laufend geändert wird.

Schutz: Speicherdesign

Das Speichermedium für die meisten Programme ist das ROM. Der Inhalt eines in der Industrie im Allgemeinen verwendeten ROMs kann mit einem Lichtmikroskop Bit für Bit gelesen werden. Diese Bits zu Bytes zusammensetzen und die Bytes zu dem kompletten ROM-Code, stellt dann keine große Schwierigkeit mehr dar. Um genau diese Analyse zu unterbinden, befindet sich das ROM nicht in den obersten und somit am leichtesten zu-

gänglichen Schichten, sondern in den unteren Siliziumschichten. Damit wird eine optische Analyse verhindert.

Würde man aber den Chip mit seiner Vorderseite auf einen Träger aufkleben und dann von seiner Rückseite her abschleifen, könnte man wiederum die ROM-Inhalte auslesen. Deshalb benutzt man bei den Chipkarten-Mikrocontrollern nur ionenimplantiertes ROM, dessen Dateninhalte weder im visuellen noch im IR- oder UV-Spektrum sichtbar sind. Dies schützt auch weitgehend vor so genanntem selektivem Ätzen. Bei diesem Verfahren wird versucht, den Halbleiter so zu ätzen, dass ROM-Inhalte wieder optisch sichtbar werden.

Schutz: Schutzschichten (*shield*)

Eine Gefahr ist die Analyse von elektrischen Potenzialen auf dem Chip während des Betriebes. Damit besteht bei genügend hoher Abtastfrequenz die Möglichkeit, Ladungspotenziale, d. h. Spannungen, auf sehr kleinen Kristallbereichen zu messen und so Rückschlüsse auf Dateninhalte des RAM während des Betriebes zu ziehen. Dies kann sehr zuverlässig durch Strom führende Metallisierungsebenen über den entsprechenden Speicherbereich oder den ganzen Chip verhindert werden. Entfernt man diese Metallschichten auf chemischem Weg, ist der Chip nicht mehr funktionsfähig, da er diese Schichten als elektrische Spannungszuführungen zur Funktion benötigt. Oft werden sogar mehrere Schutzschichten übereinander angeordnet und permanent auf Unversehrtheit geprüft.

Zusätzlich können halbleitertechnisch über die gesamte Chipoberfläche oder über besonders gefährdete Bereiche (z. B.: Unterfrequenzdetektor) mäanderförmige und Strom führende Strukturen gelegt werden. Sie lassen sich problemlos über Widerstands- oder Kapazitätsmessung messtechnisch überwachen oder in die Chipfunktion einbinden, sodass eine Beschädigung zum sofortigen Abschalten des Chips führt. Die Sicherheit kann dabei nochmals erhöht werden, wenn die Verschaltung der mäanderförmigen Strukturen während einer Sitzung geändert wird. Dies verhindert, dass die Mäander mittels FIB (*Focused Ion Beam*) überbrückt werden können.

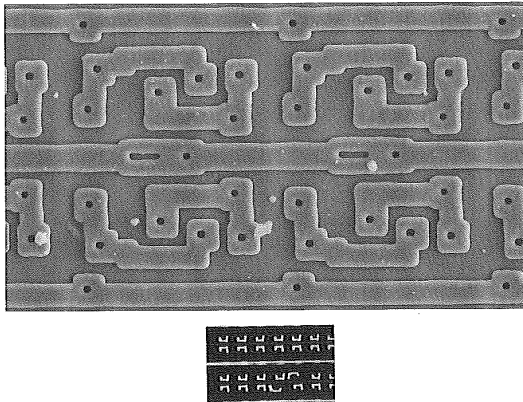


Bild 8.24 Das Foto zeigt mehrere RAM-Zellen in 3000-facher Vergrößerung, die keinen Schutz durch zusätzliche Metallisierungsebenen besitzen. Das untere Bild zeigt die elektrischen Potenziale derselben RAM-Zellen, gemessen mit einem Elektronenstrahl-Tester während des Betriebs. Man kann deutlich die Verteilung von Nullen und Einsen im RAM erkennen. (Giesecke & Devrient)

Denkbar sind auch lichtundurchlässige Schutzschichten, die mit einfach auf Halbleitern zu realisierenden Fototransistoren ständig auf Unversehrtheit geprüft werden. Sobald die Schicht abgetragen wird, würde dies erkannt und der Chip dann jeden weiteren Betrieb verweigern.

Angriff und Abwehr: Auslesen des flüchtigen Speichers

Bekanntermaßen verliert RAM seinen Dateninhalt, wenn man die dazugehörige Stromversorgung abschaltet. Dies ist jedoch nicht der Fall, wenn man die Speicherzellen auf unter $-60\text{ }^{\circ}\text{C}$ abkühlt. Speichern RAM-Zellen über längere Zeit immer nur den gleichen Inhalt, geht der RAM-Inhalt ebenfalls nicht zwangsläufig beim Abschalten der Stromzufuhr verloren. Der Hintergrund dieser Effekte ist in Peter Gutmans Papier über das sichere Löschen von Speichermedien beschrieben [Gutmann 96]. Geheime Schlüssel werden deshalb von Chipkarten nur so lange im RAM gehalten, wie sie unbedingt benötigt werden, und anschließend sofort gelöscht bzw. mit anderen Werten überschrieben. So wird das Risiko, dass geheime Schlüssel „Spuren“ in RAM-Zellen hinterlassen, minimiert, und Angriffe über Einfrieren und Einbrennen werden abgeschwächt.

RAM-Zellen auszulesen ist sehr schwierig, da dazu der Schaltungszustand der entsprechenden Transistoren detektiert werden muss. Allerdings können mit hochwertigen Elektronenmikroskopen und besonderen Kontrastverfahren durchaus gespeicherte Daten aus RAM-Zellen extrahiert werden. Voraussetzung ist allerdings die Entfernung der Passivierungsschicht und der darunter liegenden Metallisierungsebenen, die das RAM vor genau diesem Angriff schützen. Das Abtragen der Metallisierungsschichten führt aber zwangsläufig zur Zerstörung der RAM-Zellen, da diese Schichten einen Teil der Funktionalität beinhalten.

Schutz: Scrambling des Speichers

Analog zum seit langem üblichen Scrambling von Bussen findet zunehmend auch ein Scrambling der Speicher auf den Mikrocontroller-Chips statt. Die Sicherheit beruht auf der Geheimhaltung des Verwürfelungsschemas der Speicherzellen. Das Speicherscrambling ist einfach zu realisieren und benötigt wenig zusätzlichen Platz auf dem Chip. Ohne die entsprechenden Scrambling-Informationen ist es für einen Angreifer extrem schwierig, herauszufinden, wie die Speicherzellen durchadressiert sind.

Ein Speicherscrambling des EEPROMs kann auch kraft Software durchgeführt werden. Dies ist jedoch programmtechnisch aufwendig, und alle Schreibvorgänge müssen durch ato-

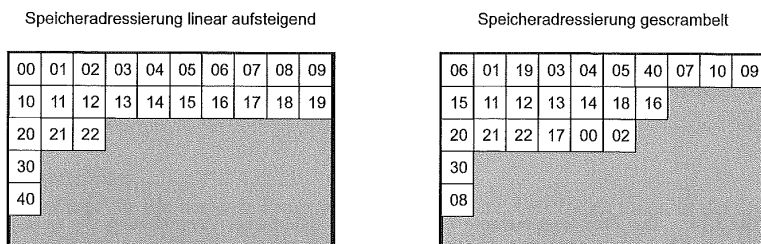


Bild 8.25 Gegenüberstellung eines in der Halbleiterindustrie üblichen Speichers zu einem geschrämbelten Speicher von Chipkarten-Mikrocontrollern.

mare Abläufe abgesichert sein, da sonst das System sehr anfällig gegenüber dem plötzlichen Abschalten der Spannung wäre. Allerdings hat ein Speicherscrambling mit Software den Vorteil, dass es chipindividuell sein kann und Speicherinhalte sogar dynamisch während einer Sitzung in den Speicherzellen umverteilt werden können.

Schutz: Verschlüsselung des Speichers

Neben dem Vertauschen von Daten im Speicher (*scrambling*) bieten moderne Chipkarten-Mikrocontroller auch diese chargen- oder chipindividuelle Verschlüsselung des Speichers und zum Teil auch der Register des Prozessors an. Dabei werden beim Lesen und Schreiben die entsprechenden Daten in Echtzeit ent- oder verschlüsselt. Zusätzlich zum Schlüssel kann bei manchen Chiptypen noch die Speicheradresse in den Ver-/Entschlüsselungsprozess einfließen, sodass gleiche Daten an unterschiedlichen Stellen im Speicher nach der Verschlüsselung unterschiedliche Werte haben. Speziell für RAM-Bereiche können auch Sitzungsindividuelle Schlüssel benutzt werden.

Falls nun durch einen erfolgreichen Angriff Daten aus dem Speicher gelesen werden könnten, würde man dazu noch zusätzlich den geheimen Schlüssel benötigen, um daraus den Klartext zu gewinnen. Dies erhöht den Aufwand für einen Angreifer erheblich, da er entweder wissen muss, an welcher Stelle dieser Schlüssel gespeichert ist, oder pauschal alle auf dem Chip vorhandenen Daten lesen muss.

Angriff und Schutz: Verschlüsselte Ablage von Geheimnissen

Dieses Exempel für einen Angriff auf physikalischer Ebene ist eigentlich ein Lehrbeispiel für eine nicht besonders gut funktionierende Abwehrmaßnahme. Die grundlegende Überlegung dazu lautet folgendermaßen: Wenn es möglich wäre, das EEPROM einer Chipkarte auszulesen, sollte zumindest verhindert werden, dass ein Angreifer auf diese Art Geheimnisse wie PIN oder Schlüssel erfahren kann. Natürlich lässt sich dagegen Vorsorge treffen. Die PIN beispielsweise muss nur mit einer Einwegfunktion verschlüsselt werden, und das Ergebnis wird als Referenz für den PIN-Vergleich im EEPROM abgespeichert. Für die Einwegfunktion könnte man kartenindividuelle Schlüssel verwenden, sodass die Referenzdaten für zwei gleiche PINs in zwei verschiedenen Karten unterschiedlich wären.

Könnte man nun die Referenzdaten im EEPROM auslesen, wäre es dem ersten Anschein nach unmöglich, aus diesen Daten die PIN abzuleiten. Ein kluger Angreifer würde diesen Weg aber gar nicht erst beschreiten. Werden die üblichen vierstelligen PINs mit Dezimalziffern verwendet, hat der Zahlenraum für die PINs die untere Grenze von "0000" und die obere Grenze von "9999". Somit beträgt die Anzahl der möglichen PINs genau 10000. Wenn nun ein Angreifer den gesamten Speicher der Chipkarte lesen kann, kennt er damit auch die Einwegfunktion und den dazugehörigen kartenindividuellen Schlüssel. Mit diesen Informationen startet er nun die Einwegverschlüsselung aller möglichen PINs. Nach durchschnittlich 5000 Versuchen hat er mit der Einwegfunktion ein Ergebnis erzeugt, das mit dem Referenzwert in der Chipkarte übereinstimmt. Damit kennt der Angreifer die PIN. Man sieht, dass in diesem Fall eine Einwegfunktion für die Abspeicherung der PIN keinen großen Nutzen bringt. Die häufige Verwendung dieses Mechanismus in der Praxis begründet sich damit, dass es viel aufwendiger ist, viele Daten aus einem Speicher auszulesen, als nur die wenigen Byte der PIN. Aus diesem Grund werden oft auch alle Schlüssel auf einer Chipkarte mit einem kartenindividuellen Schlüssel verschlüsselt abgelegt.

8.2.4.1.2 Dynamische Analysen am Chipkarten-Mikrocontroller

Schutz: Passivierungsüberwachung

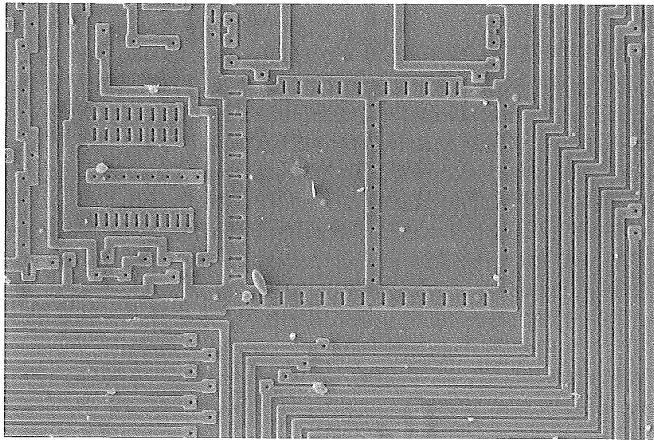


Bild 8.26 Passivierungsüberwachung in 1000-facher Vergrößerung. Der Passivierungsdetektor besteht im Wesentlichen aus den beiden rechteckigen Halbleiterelementen und basiert auf dem Prinzip der Kapazitätsmessung. (Giesecke & Devrient)

Nach der Herstellung des Mikrocontrollers auf dem Silizium wird eine Passivierungsschicht aufgetragen, die Oxidation, beispielsweise durch Luftsauerstoff, oder weitere chemische Prozesse auf der Chipoberfläche verhindert. Um Manipulationen auf dem Chip vorzunehmen, muss immer als Erstes diese Passivierungsschicht entfernt werden. Es sollte jedoch bedacht werden, dass sich die Passivierungsschicht zwar chemisch abtragen lässt,

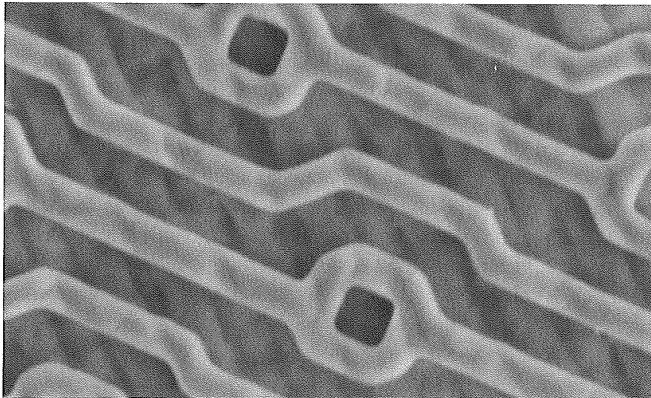


Bild 8.27 Moderne Passivierungsüberwachung in 8000-facher Vergrößerung. Der Abstand der Leiterbahnen zueinander beträgt $4\text{ }\mu\text{m}$ und das Funktionsprinzip dieses Passivierungsdetektors basiert auf dem Prinzip der Widerstandsmessung. (Giesecke & Devrient)

der Chip aber dann einer großen Oxidationsgefahr unterliegt, die ihn relativ schnell zerstören kann. Eine Sensorschaltung kann über Widerstands- oder Kapazitätsmessung feststellen, ob diese Passivierungsschicht noch vorhanden ist. Ist sie nicht mehr existent oder beschädigt, kann entweder ein Interrupt in der Chipsoftware ausgelöst werden, oder der gesamte Chip wird von der Hardware abgeschaltet, was alle dynamischen Analysen zuverlässig verhindert.

Schutz: Spannungsüberwachung

Auf jedem Chipkarten-Mikrocontroller ist eine Spannungsüberwachung vorhanden. Diese sorgt für ein definiertes Abschalten des Bausteins, wenn die oberen oder unteren Grenzen der Betriebsspannung über- bzw. unterschritten werden. Auf diese Weise erhält die Software die Sicherheit, dass ein Betrieb in den Grenzbereichen, in denen der Chip nicht mehr voll funktionsfähig ist, unmöglich ist. Wäre keine Spannungsüberwachung vorhanden, kann es in diesen Grenzbereichen vorkommen, dass z. B. der Programmzähler des Prozessors nicht mehr stabil läuft, was beispielsweise zu unkontrollierten Sprüngen innerhalb des Programms führt oder schlichtweg zu Rechenfehlern im Prozessor. Dieses Fehlverhalten lässt sich als Ansatzpunkt zur Ermittlung von geheimen Schlüsseln mittels der weiter unten im Text beschriebenen differenziellen Fehleranalyse (DFA) nutzen.

Aus diesem Grund ist es wichtig, dass die Spannungsüberwachung auch sehr kurze Spannungsspitzen oder Spannungseinbrüche erkennt, um damit typische Angriffe durch bewusste Fehlereinstreuung des Prozessors abzuwehren. Bei einer Chipkarte für den Spannungsbereich 3 – 5 V sind beispielsweise die üblichen Grenzwerte zur Abschaltung 2,3 V und 6,3 V. Diese Werte liegen etwas außerhalb des durch diverse Normen festgelegten Spannungsbereichs von 2,7 V und 5,5 V, um mögliche Schwankungsbreiten der Sensorkalibrierung bei der Halbleiterfertigung auszuschalten.

Gerade die Spannungsüberwachung ist von sehr großer Bedeutung für die Sicherheit des Mikrocontrollers. Man könnte sich hier als Angriff vorstellen, dass in einem ersten Schritt die entsprechenden Detektoren z. B. mittels FIB (*Focused Ion Beam*) unbrauchbar gemacht werden und dann in einem zweiten Schritt der eigentliche Angriff startet. Aus diesem Grund sind oft die für die Sicherheit eines Mikrocontrollers wichtigen Komponenten besonders geschützt, sodass eine Manipulation erkannt wird und sich die Chipkarte automatisch deaktiviert.

Ein Sensor, der zum Teil auf der Spannungsdetektion aufbaut, ist die Power-On-Erkennung. Diese ebenfalls in allen Chips vorhandene Detektion eines Power-On unabhängig vom Reset-Signal sorgt dafür, dass der Chip beim Anschalten immer in einen definierten Bereich gesetzt wird. Die Gründe dafür sind analog denen der Spannungsüberwachung.

Schutz: Frequenzüberwachung

Die Taktversorgung der Chipkarte läuft immer extern, sodass die interne Rechengeschwindigkeit völlig von außen bestimmt wird. Damit besteht zumindest theoretisch die Möglichkeit, den Mikrocontroller im Einzelschrittbetrieb zu fahren. Dies würde zu hervorragenden Analysemöglichkeiten vor allem in der Messung von Stromaufnahme (*Power Analysis*) und elektrischen Potenzialen auf dem Chip führen. Um diesen Angriff zu verhindern, ist auf dem Chip eine Funktionsbaugruppe zur Unter- und Überfrequenzdetektion integriert. Diese unterbindet, dass der angelegte Takt in unzulässiger Weise erniedrigt werden kann. Der Minimalwert für die Taktversorgung nach den meisten Spezifikationen liegt bei 1 MHz. Da eine Unterfrequenzdetektion technisch bedingt mit großen Streuungen verbunden ist, schaltet der

Chip üblicherweise erst im Bereich um 500 kHz ab. Damit ist sichergestellt, dass der Chip auf jeden Fall mit dem minimalen Wert von 1 MHz betrieben werden kann. Die maximale Frequenz nach den meisten Spezifikationen beträgt 5 MHz und typische Überfrequenzdetektoren schalten aus obigen Gründen bei ca. 7 MHz die Chipkarte ab. Die Hardware ist heute oft so aufgebaut, dass der Chip bei zu hoher Frequenz nicht mehr betrieben werden kann.

Zum Schutz vor dem gefährlichen Einzelschrittbetrieb des Mikrocontrollers ist es natürlich notwendig, den Unterfrequenzdetektor durch Schutzschichten abzusichern, sodass sich dieser keinesfalls unbemerkt manipulieren lässt.

Schutz: Temperaturüberwachung

Eine Sensorvariante, die Temperaturüberwachung, wird zwar bei manchen Chiptypen eingesetzt, doch ist ihr Zweck umstritten. Eine kurzzeitige Temperaturerhöhung über oder unter dem Spezifikationsbereich führt noch zu keiner dauerhaften Schädigung des Chips und stellt auch keinen Angriff dar. Ein Abschalten in dieser Grenzsituation könnte eher zu einer künstlich erhöhten Ausfallrate führen und bringt dem Betreiber eines Chipkarten-Systems keine zusätzliche Sicherheit.

Schutz: Scrambling der Busse

Viele Chipkarten-Mikrocontroller scambeln die nur intern auf dem Chip zugänglichen Busse zur Ansteuerung der Speicher. Dies bedeutet, dass die einzelnen Busleitungen nicht in auf- oder absteigender Folge, sondern wirr und mehrfach gegeneinander vertauscht

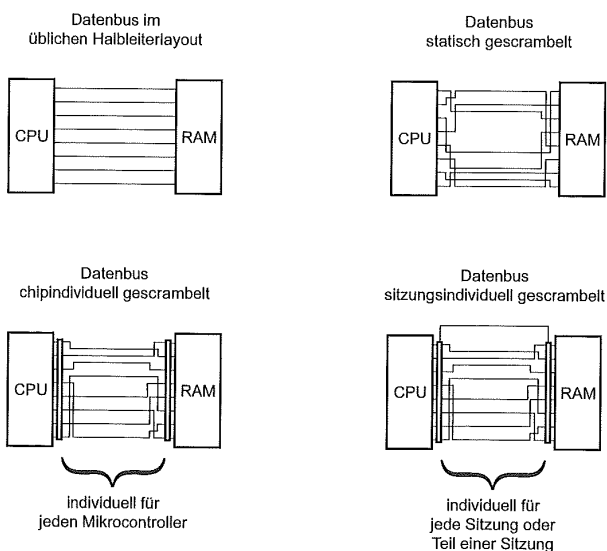


Bild 8.28 Darstellung des Scrambling von Bussen in Chipkarten-Mikrocontrollern anhand des 8 Bit breiten Datenbusses zwischen CPU und RAM. Die Linien des Datenbusses sollen Informationsflüsse darstellen und keine elektrische Leitungen. Die Verschlüsselungseinheiten sind nur der Übersichtlichkeit wegen als separate Komponenten dargestellt. In der Wirklichkeit sind sie mit den restlichen Komponenten auf eine Weise verflochten, dass sie nicht mehr als separate Einheiten erkenn- und somit angreifbar sind.

nebeneinander oder sogar getrennt durch Isolationsschichten übereinander angeordnet sind. Für den potenziellen Angreifer ist dies eine zusätzliche Hürde, da er nicht weiß, welche Busleiterbahn welche Funktion bzw. Adresse hat.

Dieses Mischen der Leiterbahnen wurde als ursprünglich nur in einer statischen Variante eingeführt, d. h. auf jedem Chip sind die Vertauschungen identisch. Damit wäre es wahrscheinlich mittelfristig für einen Angreifer kein größeres Problem, die Vertauschung herauszufinden und bei einer Abhöraktion entsprechend zu berücksichtigen.

Es gibt hier aber eine Verbesserung der Sicherheit, indem man ein chipindividuelles Scrambling der Busse einführt. Dieses chipindividuelle Scrambling kommt natürlich nicht zustande, indem man für jeden Chip eigene Belichtungsmasken für die Busse erstellt, da dies technisch momentan nicht realisierbar bzw. unbezahlbar wäre. Das Scrambling wird dabei durch direkt an den Speichern sitzende Verwürfler vorgenommen, die beispielsweise über die chipindividuelle Nummer angesteuert werden. Dies ist halbleitertechnisch ohne großen Aufwand möglich und erschwert einem Abhörer das Leben beträchtlich. Ein chipindividuelles und sitzungsspezifisches Scrambling ist durch variable Eingangswerte der Verwürfler somit ebenfalls realisierbar.

Schutz: Irreversible Umschaltung vom Testmodus in den Benutzermodus

Zur Überprüfung der Chips während der Halbleiterfertigung und zur Ausführung der internen Testprogramme besitzen alle Mikrocontroller einen so genannten Testmodus, in dem die Halbleiter noch auf dem Wafer oder im Modul beim Hersteller geprüft werden können. Dieser Testmodus erlaubt jedoch Zugriffsarten auf den Speicher, die später strikt verboten sind. Aus fertigungstechnischen Gründen ist es leider zwingend erforderlich, dass in diesem Modus auch Daten aus dem EEPROM gelesen werden können.

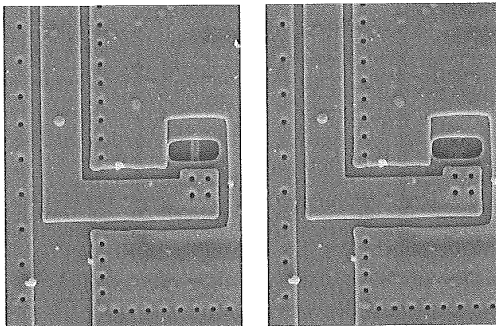


Bild 8.29 Polysilizium-Sicherung (*fuse*) in 2000-facher Vergrößerung. Das linke Bild zeigt die Sicherung in nicht gebranntem Zustand und das rechte Bild in gebranntem Zustand. (Giesecke & Devrient)

Die Umschaltung vom Testmodus in den Benutzermodus muss irreversibel sein. Dies kann mit Polysilizium-Sicherungen auf dem Chip bewerkstelligt werden. Dazu wird an einem dafür vorgesehenen Testpunkt auf dem Siliziumkristall eine Spannung angelegt, und diese führt dazu, dass die betreffende Sicherung durchbrennt. Damit ist der Chip per Hardware in den Benutzermodus umgeschaltet. Dies kann im Regelfall nicht mehr rück-

gängig gemacht werden. Aufgrund der Funktionsweise der Sicherung handelt es sich jedoch um eine verhältnismäßig große Struktur auf der Chipoberfläche. Es ist vorstellbar, nach teilweiser Entfernung der Passivierungsschicht über der Sicherung diese mechanisch zu überbrücken. Dadurch wäre der Mikrocontroller wieder im Testmodus, und mit den erweiterten Zugriffsmöglichkeiten für Tests ließe sich der Speicher auslesen. Würde der komplette Speicherinhalt bekannt, könnte man problemlos die ausgelesene Chipkarte klonen.

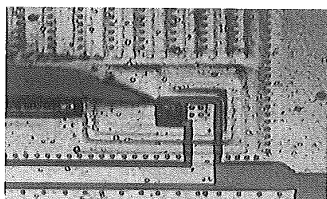


Bild 8.30 Polysilizium-Sicherung (*fuse*) mit einer Mikroprobenadel in 500facher Vergrößerung. Mit der Mikroprobenadel könnte eine gebrannte Sicherung überbrückt werden. (Giesecke & Devrient).

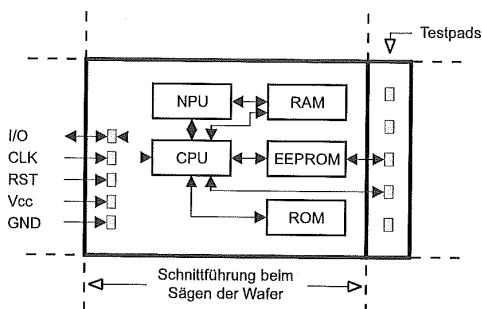


Bild 8.31 Darstellung einer von mehreren Varianten der irreversiblen Trennung der Testpads für CPU- und Speichertest eines Chipkarten-Mikrocontrollers.

Um diesen Angriff abzuwehren, gingen die meisten Halbleiterhersteller dazu über, zusätzlich zur Sicherung aus Silizium einen Bereich im EEPROM für den Umschaltmechanismus zu reservieren. Steht dort ein bestimmter und nicht mehr veränderbarer Wert, ist der Chip irreversibel in den Benutzermodus geschaltet. Selbst nach Überbrückung der Sicherung befindet sich der Chip nicht wieder im Testmodus, weil dies der zusätzliche logische Schalter im EEPROM verhindert.

Eine weitere Steigerung der Sicherheit des Umschaltens vom Testmodus in den Benutzermodus lässt sich durch eine sehr einfache konstruktive Maßnahme beim Aufbau des Mikrocontrollers auf dem Wafer durchführen. Die zur Kontaktierung für das Testing notwendigen Testpads werden beim Sägen des Wafers in einzelne Dice einfach abgeschnitten. Damit braucht man keine Sicherung und keine EEPROM-Zellen für den Umschaltvorgang, da die für den Testmodus notwendigen Elemente nicht mehr vorhanden sind. Es ist natürlich auch möglich, die Sicherung zur Umschaltung von Test- in Benutzermodus durch eine

Leiterbahn zu ersetzen, welche beim Sägen des Wafers irreversibel getrennt wird. Die Kontaktierung an den abgeschnittenen Leiterbahnen an der Seite des Chips ist mit heutiger Technologie nicht möglich.

Dynamische Analyse und Abwehr:

Abhören der Busse zu den Speichern des Mikrocontrollers

Um die Busse auf dem Mikrocontroller zwischen CPU und den Speichern (ROM, EEPROM, RAM) abzuhören, muss zuerst der Chip freigelegt und dann die auf dem Chip befindliche Passivierung entfernt werden. Die Passivierung schützt einerseits die Chipoberfläche vor Oxidation und andererseits vor Angriffen, da ihre Unversehrtheit über Sensoren überwacht wird. Nach Anderson und Kuhn [Anderson 96b] kann sie durch Ätzen mit Flusssäure vom Chip entfernt werden. Außerdem ist auch ein Lasercutter¹⁰ gut geeignet, um selektiv nur an den benötigten Stellen Aussparungen in die Passivierung zu bohren.

Nach Entfernen der Passivierung von der gesamten Chipoberfläche oder Teilen davon wäre es zumindest vorstellbar, den Adress-, Daten- und Steuerbus für die Speicher mit Mikroprobenadeln zu kontaktieren. Falls zu allen Leitungen der drei Busse eine elektrische Verbindung hergestellt werden kann, ist es sehr einfach, die einzelnen Speicherzellen zu adressieren und bei ROM und EEPROM beliebige Bereiche auszulesen. Dabei muss der Chip nicht angeschaltet sein und kann sich in einer beliebigen Kontaktiervorrichtung befinden. Die Auswirkungen eines erfolgreichen Angriffs mit dieser Methode sind gravierend, da damit alle geheimen Daten aus den nichtflüchtigen Speichern prinzipiell lesbar werden.

Dieses Verfahren ließe sich noch erweitern, indem man die Busse kontaktiert und dann den Chip ganz normal betreibt. Dann könnte man den gesamten Datenverkehr zwischen CPU und den Speichern abhören und mit einem genügend schnellen Logikanalysator aufzeichnen.

Wie schon angedeutet, ist es sehr schwierig, mit einzelnen Leiterbahnen auf dem Chip eine elektrische Verbindung herzustellen. Die Anzahl der notwendigen Verbindungen bei einem 8-Bit-Mikrocontroller bei diesem Angriff beträgt 16 für den Adressbus, 8 für den Datenbus und 1 bis 4 für den Steuerbus. Zusammengerechnet müssten also gleichzeitig mindestens 25 Verbindungen zwischen einem externen Analysecomputer und den Leiterbahnen auf dem Chip hergestellt werden. Dies ist auch mit moderner Mikromanipulationstechnik durch die sehr kleinen Strukturen auf dem Halbleiter momentan noch nicht zu bewerkstelligen. Es wäre allerdings möglich, mit einem in der Halbleiterindustrie üblichen Werkzeug, einem fokussierten Ionenstrahl (FIB – *focused ion beam*), an die Busse eine Art von elektrisch leitenden Kontaktflächen anzubringen, welche dann als Aufsetzpunkte für Mikroprobenadeln genutzt werden könnten. Der Aufwand dafür ist allerdings enorm.

Sollte es aber gelingen, müsste vor einem erfolgreichen Auslesen auch noch das Scrambling der Busse ermittelt werden, da sich die einzelnen Leiterbahnen nicht geordnet nebeneinander auf dem Chip befinden, sondern in einer von außen nicht erkennbaren Anordnung.

¹⁰ Ein Lasercutter ist eine Vorrichtung zum Bohren und Schneiden mit einem energiereichen Laserstrahl in einer Genauigkeit von Teilen eines Mikrometers.

Wenn es in einigen Jahren mit stark verbesserter Technik möglich sein wird, die Busse heutiger Mikrocontroller zu kontaktieren, hätte dies wahrscheinlich keine Auswirkungen auf die Sicherheit, denn bis dahin sind die Strukturen auf den Halbleitern noch wesentlich feiner, als es heute schon der Fall ist. Zudem wird die Mikromechanik wohl immer der auf optischen Verfahren beruhenden Halbleitertechnik hinterherhinken. Damit sind Angriffe nach dieser Vorgehensweise wahrscheinlich auch in Zukunft nicht geeignet, die Sicherheit einer Chipkarte signifikant zu beeinträchtigen.

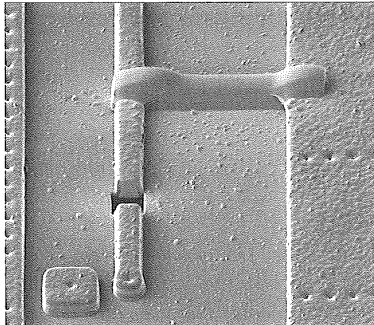


Bild 8.32 Anwendung eines Focused Ion Beam (FIB) an einem Halbleiterchip. Die auf der Oberfläche des Chips von oben nach unten verlaufende Leiterbahn wurde mithilfe eines FIB durchtrennt. Zusätzlich wurde die getrennte Leiterbahn durch eine mittels FIB neu abgeschiedene Metallstruktur mit einer parallel dazu verlaufenden Leiterbahn kontaktiert, was in der oberen Bildhälfte zu sehen ist.

(Quelle: Fraunhofer Institut für Integrierte Schaltungen, Bereich Bauelementetechnologie)

Dynamische Analyse und Abwehr: Messen des Stromverbrauchs der CPU

Schon 1995 hieß es in der ersten Auflage dieses Buches an dieser Stelle: „Auch das Design des Prozessors ist für die Sicherheit ausschlaggebend. Ein Chipkarten-Prozessor muss für alle Maschinenbefehle eine annähernd gleiche Stromaufnahme haben. Ist dies nicht der Fall, kann aufgrund des Stromverbrauches auf den gerade abgearbeiteten Befehl geschlossen werden. Daraus ließe sich wiederum einiges an geheimer Information ableiten.“ Es war also eine seit Jahren bekannte Tatsache, dass man aufgrund des Stromverbrauchs eines Prozessors auf dessen eben ausgeführten Befehl und sogar auf die dabei verarbeiteten Daten schließen konnte, als im Juni 1998 Paul Kocher, Joshua Jaffe und Benjamin Jun ein Dokument über einfache Leistungsanalyse (*simple power analysis – SPA*) und differenzielle Leistungsanalyse (*differenzial power analysis – DPA*) veröffentlichten [Kocher 98a]¹¹.

Das Prinzip der einfachen Leistungsanalyse (SPA) ist relativ einfach. Mit einem Analog-Digital-Wandler wird anhand des Spannungsabfalls an einem seriell vorgeschalteten Widerstand mit hoher zeitlicher Auflösung der Stromverbrauch eines Mikrocontrollers gemessen. Bei Hochleistungsprozessoren wie Pentium oder PowerPC könnte aufgrund der komplexen internen Abläufe nicht mehr auf die ausgeführten Befehle geschlossen werden. Der relativ einfache Aufbau der 8051- und 6805-CPU's von Chipkarten-Mikrocontrollern führt jedoch

¹¹ Ein detaillierterer Überblick zu diesem Thema findet sich bei [Kocher 98b] und [Messerges 99].

dazu, dass die internen Abläufe und verarbeiteten Daten zu messbaren und auch interpretierbaren Auswirkungen auf den Stromverbrauch führen. Zur Verdeutlichung kann man sich vorstellen, dass der gleiche Programmablauf mit gleichen Daten zu einem bestimmten zeitlichen Verlauf des Stromverbrauchs des Prozessors führt. Wird dieses Programm nun mit anderen Daten durchlaufen, unterscheidet sich der zeitliche Verlauf des Stromverbrauchs. Diese Abweichung verwendet man nun zur Ermittlung der in dem Ablauf prozessierten Daten.

Mit der differentiellen Leistungsanalyse (DPA) können gegenüber der einfachen Leistungsanalyse (SPA) noch geringere Unterschiede beim Stromkonsum des Mikrocontrollers aufgedeckt werden. Dazu misst man den Strom zuerst bei der Abarbeitung bekannter und anschließend unbekannter Daten. Die Messungen werden in der Regel viele Male wiederholt, damit durch Mittelwertbildung das Rauschen eliminiert werden kann. Im Anschluss an diese Messungen bildet man die Differenz und kann von diesem Ergebnis auf die unbekannten Daten schließen.

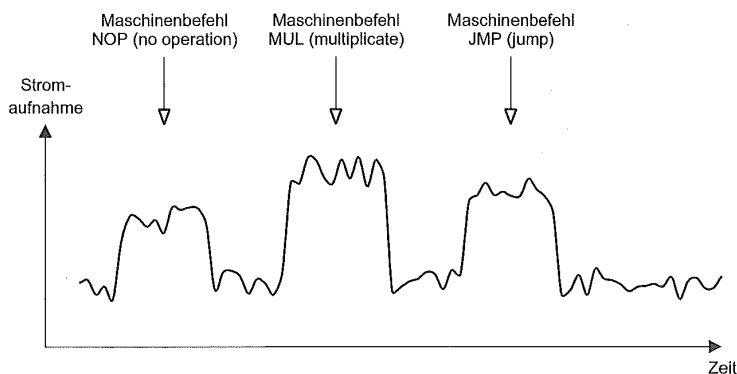


Bild 8.33 Vereinfachte Prinzip-Darstellung der Schwankungen im Stromverbrauch bei der Ausführung von Maschinenbefehlen eines Chipkarten-Mikrocontrollers. Der Stromverbrauch kann neben einer Abhängigkeit vom jeweiligen Maschinenbefehl noch zusätzlich eine Abhängigkeit von den jeweils verarbeiteten Daten besitzen.

In dem Papier von Kocher et. al. ist als Weiterentwicklung der DPA noch die High-Order Differenzial Power Analysis (HO-DPA) erwähnt. Dabei misst man nicht alleine den Stromverbrauch des Mikrocontrollers, sondern zusätzlich im Prozessor ausgeführte programmabhängige Größen wie beispielsweise die elektromagnetische Abstrahlung des Chips. Die so gesammelten Messwerte von bekannten und unbekannten Daten benutzt man dann analog der DPA zur Differenzbildung und kann dann daraus die geheimen Daten errechnen.

Die drei Varianten der Leistungsanalyse für Chipkarten-Mikrocontroller sind für unvorbereitete Hard- und Software sehr ernst zu nehmende Angriffe. Der Grund dafür besteht darin, dass bei manchen Mikrocontrollern durchaus Abhängigkeiten des Stromverbrauchs vom jeweiligen Maschinenbefehl und auch von den in diesem Maschinenbefehl verarbeiteten Daten bestehen. Zudem hält sich der für einen erfolgreichen Angriff erforderliche Aufwand an Messgeräten in Grenzen. Es existiert aber eine Reihe wirkungsvoller Gegenmaßnahmen, die sich einerseits auf entsprechend verbesserte Hardware und andererseits auf modifizierte Software stützen.

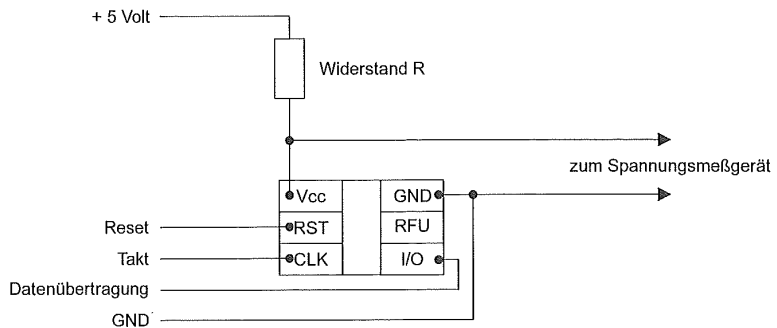


Bild 8.34 Die Beschaltung eines Chipkarten-Mikrocontrollers, um auf einfache Weise unter Verwendung eines Vorwiderstands Strommessungen durchzuführen.

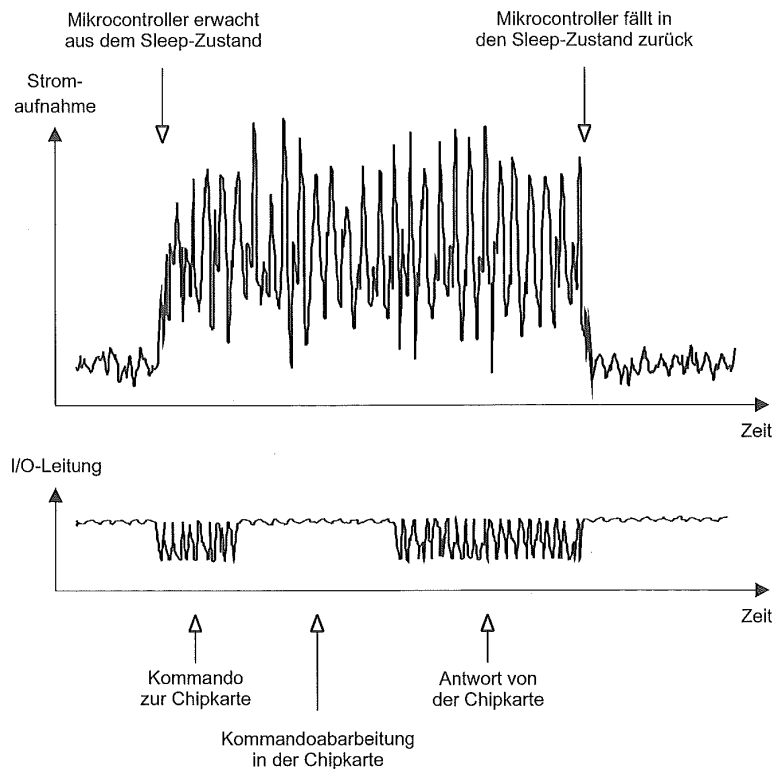


Bild 8.35 Überblickshaften Darstellung des Stromverbrauchs eines Chipkarten-Mikrocontrollers im Ruhezustand und der Stromschwankungen während des Betriebs. Man kann anhand des aufgenommenen Stromes erkennen, wie der Mikrocontroller durch die erste fallende Flanke auf der I/O-Leitung aus dem Sleep-Zustand erwacht und anschließend in Abhängigkeit der ausgeführten Maschinenbefehle einen laufend differierenden Strombedarf hat.

Die einfachste Hardware-technische Lösung ist der Einbau eines schnellen Spannungsreglers auf dem Chip, der über einen Shunt-Widerstand einen von Maschinenbefehl und Daten unabhängigen Stromverbrauch sicherstellt. Künstliche Stromrauschquellen auf dem Chip sind ebenfalls eine wirksame Lösung. Eine technisch aufwendigere Lösung besteht in einem modifizierten halbleitertechnischen Design des Prozessors, damit dieser immer einen konstanten Stromverbrauch aufweist. Diese Lösungsansätze erhöhen jedoch zum Teil den Stromverbrauch des Mikrocontrollers, was in bestimmten Anwendungsfeldern wie der Telekommunikation nicht gewünscht ist. Eine einfache Abwehrmaßnahme kann auch darin bestehen, während eines SPA/DPA-kritischen Vorgangs für diesen Vorgang nicht benötigte Komponenten wie CRC-Prüfsummengenerator oder den numerischen Coprozessor mit Zufallsdaten als Eingangswerte zu aktivieren, um auf diese Weise ein künstliches Rauschen des Stromverbrauchs zu erzeugen.

Die Verwendung von zufallsgesteuerten Wartezeiten (*random wait states*) beim Prozessor erschwert die Synchronisation bei der Stromanalyse erheblich, ohne dabei die Nachteile eines erhöhten Stromverbrauchs in Kauf zu nehmen. Ein ähnlicher Lösungsansatz wird bei Chipkarten-Mikrocontrollern mit eigener Takterzeugung auf dem Chip verfolgt, indem die Taktfrequenz zufallsgesteuert innerhalb vorgegebener Grenzen permanent variiert wird.

Bei den softwaretechnischen Gegenmaßnahmen gibt es mittlerweile eine immense Bandbreite an Lösungsvarianten. Im Folgenden seien einige wenige davon stellvertretend kurz aufgeführt. Der einfachste Ansatz ist die ausschließliche Benutzung von Maschinenbefehlen mit einem sehr ähnlichen Stromverbrauch. Maschinenbefehle, welche eine signifikante Abweichung vom durchschnittlichen Strombedarf haben, dürfen dann nicht mehr im Assemblercode verwendet werden. Eine weitere Lösungsmöglichkeit besteht darin, für gleiche Berechnungen in Kryptoalgorithmen verschiedene Abläufe einzuführen, welche jeweils zufällig ausgewählt werden. Dies erschwert es dem Beobachter erheblich, eine Konvergenz zwischen bekannten und unbekannten Maschinenbefehlen sowie verarbeiteten Daten zu erkennen. Eine Abwehr in ähnlicher Richtung wird mit chipindividuellen Tabellen für die S-Boxen des Triple-DES-Algorithmus verfolgt.

Um die zu einer erfolgreichen Stromanalyse vorab benötigte Datenerfassung zu erschweren, sollten alle Schlüssel mit irreversibel ablaufenden Fehlbedienungsanzählern abgesichert sein. Weiterhin ist es notwendig, den freien Zugriff auf alle Kommandos in der Art von INTERNAL AUTHENTICATE, bei denen beliebige Daten durch einen Kryptoalgorithmus der Chipkarte geschickt werden können, zu sperren. Müssen Kommandos mit dieser Eigenschaft aus bestimmten Gründen zwingend verwendet werden, so muss die Chipkarte vorab das Terminal auf Echtheit prüfen. Durch diese Einschränkung der Kommandobnutzung verhindert man ebenfalls die Sammlung von Referenzdaten für die spätere Stromanalyse.

Grundsätzlich ist anzumerken, dass keine geheimen Daten bitweise vom Prozessor verarbeitet werden sollen, da dies die SPA/DPA-Analyse erheblich vereinfacht. Sind Schlüssel in die Register eines Kryptocoprozessors zu laden, so werden bei manchen Realisierungen vermischt dazu Zufallszahlen als Dummywerte in die entsprechenden Register geladen, sodass entsprechende Messungen ins Leere laufen. Am Ende eines solchen Ladevorgangs müssen selbstverständlich die echten Schlüssel in den Registern stehen.

SPA/DPA-Analysen lassen sich jedoch nicht nur zur Ausspähung von in Chipkarten gespeicherten geheimen Daten benutzen. Sie können auch verwendet werden, um beispielsweise zweifelsfrei nachzuweisen, dass ein bestimmter Programmcode in einer Chipkarte

verwendet wird. Dazu führt man bei der zu prüfenden Chipkarte eine SPA-Analyse der betreffenden Funktion durch und vergleicht die gewonnenen Stromverlaufskurve mit der einer Referenzkarte. Damit kann man unter günstigen Umständen und ohne Wissen um den verwendeten Quellcode den Nachweis erbringen, das beispielsweise in einem Konkurrenzprodukt Teile von fremden Programmcode benutzt werden. Der technische Hintergrund dabei ist, dass bei gleichem Quellcode in der Regel vom gleichen Compiler auch gleicher Maschinencode erzeugt wird. Beim anschließenden Linkvorgang treten dann wegen der sicherlich unterschiedlichen Lokalisierung im Speicher noch kleinere Abweichungen auf.

Die Prüfung von Software in Chipkarten auf Resistenz gegen SPA/DPA-Angriffe hat mittlerweile ein hohes Niveau erreicht und in der Folge durchaus den Charakter einer Spezialdisziplin erlangt. Es ist dabei üblich geworden, dass parallel zur Softwareentwicklung periodisch Messungen stattfinden und entsprechend den Messergebnissen die Software adaptiert wird, sodass SPA/DPA-Angriffe ins Leere laufen würden. Am Anfang einer Software-Entwicklung führt man die Messungen mit im EEPROM befindlicher Software durch und kann die Analysen dann erweitern, wenn sich bei Erhalt der ersten Muster vom Halbleiterhersteller die Software im ROM des Mikrocontrollers befindet. Durch Erfahrung hat man nämlich feststellen müssen, dass dieser Aspekt bei SPA/DPA-Messungen durchaus von Belang ist.

Bei SPA und DPA können naturgemäß nicht nur Angriffe gegen Kryptoalgorithmen getätigt werden. Beide Methoden eignen sich auch sehr gut zur Analyse aller Aktivitäten des Prozessors. Entsprechende Erfahrung und Ausrüstung vorausgesetzt, können mit SPA und DPA bei einer dagegen nicht resistenten Chipkarte selbst die Daten von Kopiervorgängen im Speicher ermittelt werden.

Analyse und Abwehr:

Messen der elektromagnetischen Abstrahlung der CPU

Zumindest theoretisch könnte man analog zur differenziellen Leistungsanalyse anhand der elektromagnetischen Abstrahlung Schlüsse auf interne Abläufe auf dem Chipkarten-Mikrocontroller ziehen. Mit SQUIDS (*superconducting quantum interference devices*) lassen sich Magnetfelder von geringer Ausdehnung und Stärke messen. Die Auswertung kann dann analog SPA/DPA erfolgen. Allerdings ist der nötige technische Aufwand enorm und das zwingend erforderliche Wissen über die internen Strukturen des Halbleiters nicht allgemein verfügbar. Zudem können Halbleiterbausteine gegen diese Art von Angriffen sehr effektiv geschützt werden, indem man mehrere Leiterbahnen übereinander legt, sodass zwar ein Magnetfeld mit empfindlichen Detektoren gemessen werden kann, aber nicht, welche der übereinander liegenden Leitungen Strom führend ist.

8.2.4.1.3 Manipulationen am Chipkarten-Mikrocontroller

Manipulation und Abwehr:

Ändern von Speicherinhalten am Chipkarten-Mikrocontroller

Das direkte Lesen von Speicherinhalten eines Mikrocontrollers ist ein mögliches Angriffsszenario, dessen Gefährlichkeit auf den ersten Blick zu erkennen ist. Ein ähnliches Szenario, bei einer nahezu gleichen Angriffsstärke, ist das bewusste Ändern von Dateninhalten in einem Speicher eines Chipkarten-Mikrocontrollers. Damit ist nicht gemeint, zufällige Fehler im Berechnungsvorgang eines Kryptoalgorithmus einzustreuen, wie dies als Ansatz

für die differenzielle Fehleranalyse (DFA) benutzt wird, sondern gezielt bestimmte Bits oder Bytes im ROM oder EEPROM zu ändern.

Ungezielte Änderungen in allen Speicherarten lassen sich beispielsweise durch Röntgenstrahlung auf das Modul oder UV-Licht auf den freigelegten Chip zuwege bringen. So können mit Bestrahlung durch UV-Licht EEPROM-Zellen entladen werden, und ihr Speicherinhalt nimmt dann den Wert des energieärmsten Zustands an. Dieser Vorgang ist vollkommen analog zum üblichen Löschen von EPROMs unter einer UV-Lampe. Für einen Angriff lässt sich dies aber nicht sinnvoll nutzen, da der Angreifer keine Kontrolle darüber hat, welche EEPROM-Zellen gekippt werden.

UV-Strahlung kann aber gebündelt oder durch einen Laser mit einem fein fokussierten Strahl ersetzt werden. Dann könnten durchaus einzelne Speicherzellen geändert werden. Der Laser hat den Vorteil, dass sich damit bei genügender Leistung auch ROM-Speicher ändern lassen. Ähnlich lässt sich auch ein fokussierter Ionenstrahl (*focused ion beam*) zur Änderung von Speicherzellen benutzen.

Die damit möglichen Änderungen können durchaus für theoretisch effektive Angriffe genutzt werden. So könnte beispielsweise der Zufallszahlengenerator so manipuliert werden, dass er keine Zufallszahlen mehr liefert, sondern immer den gleichen Wert. Wäre dies möglich, kann die Authentisierung des Terminals durch die Chipkarte durch Wiedereinspielung eines alten Wertes (Replay-Angriff) gebrochen werden.

Durch bewusstes und bitgenaues Ändern von Speicherinhalten kann man sich aber durchaus noch andere Angriffe vorstellen. So könnten beispielsweise alle S-Boxen des DES-Algorithmus gezielt einheitlich auf null oder eins abgeändert werden. Damit ist der DES kein Verschlüsselungsalgorithmus mehr, sondern lediglich eine lineare Transformation [Anderson 96 a].

Kennt man die exakte Position der DES-Schlüssel im EEPROM und ist es möglich, diese bitweise zu verändern (z. B.: durch fokussierte UV-Strahlung), kann man sich dies natürlich ebenfalls für einen wirkungsvollen Angriff zunutze machen. Dazu setzt man ein beliebiges Bit im Schlüssel auf null und ruft anschließend ein Kommando auf, das den DES-Algorithmus mit dem veränderten Schlüssel benutzt. Erhält man einen Returncode, der einen Paritätsfehler im Schlüssel anzeigt, so war das geänderte Bit auf eins gesetzt; falls kein Paritätsfehler gemeldet wird, stand es bereits auf null. Anschließend verfährt man sinngemäß mit den restlichen 55 Bits des Schlüssels, und als Ergebnis erhält man den geheimen Schlüssel [Zieschang 98].

Es ließen sich noch viele weitere Angriffe aufführen, die in eine ähnliche Richtung laufen – wie gezielte Veränderung von Programmabläufen oder Umsetzen von Zeigern. So interessant und einfach alle diese Attacks aber auf dem Papier aussehen, so schwierig wäre es, sie in der Realität durchzuführen. Die notwendigen Voraussetzungen für einen erfolgreichen Angriff sind nicht gerade simpel und führen dazu, dass dieser Angriff ein interessantes, aber doch theoretisches Gebilde bleibt.

Um Bits gezielt verändern zu können, muss der Angreifer ein detailliertes Wissen über die physikalischen Adressen von Daten und Programmcode im Speicher besitzen und außerdem das Scrambling und/oder Verschlüsselung des jeweiligen Speichers kennen. Zudem sind alle sicherheitskritischen Daten- und Programmcodeteile über Prüfsummen abgesichert, welche vor einer Benutzung jeweils nachgerechnet werden. Ein Angreifer müsste also auch noch gezielt die Prüfsummen an die veränderten Daten anpassen. Vergessen

werden darf auch nicht, dass vor einer Manipulation alle Schutzschichten über dem entsprechenden Speicher auf dem Chip zu neutralisieren sind. Alle diese Punkte führen dazu, dass die Attraktivität für diese Form eines Angriffs gegen null geht, obwohl sie sich zugegebenermaßen in der Theorie sehr attraktiv anhört.

8.2.4.2 Angriffe auf der logischen Ebene

Angriffe auf die Sicherheit einer Chipkarte auf logischer Ebene setzen vor allem ein Verständnis der Kommunikation und des Informationsflusses zwischen Terminal und Chipkarte voraus. Es ist nicht so sehr notwendig, Prozesse auf der Hardware-Ebene zu verstehen, wie die Abläufe in der Software. Die hier beispielhaft vorgestellten Szenarien bewegen sich informationstechnisch gesehen eine Ebene über den Angriffen, bei denen vor allem die Hardwareeigenschaften ausgenutzt werden.

Angriff und Abwehr: Dummy-Chipkarte

Der wohl am ehesten vorstellbare Angriff ist die Verwendung einer selbst programmierten und mit diversen Analyse- und Protokollfunktionen erweiterten Chipkarte. Bis vor einigen Jahren war dies so gut wie undurchführbar, da der Erwerb von Chipkarten bzw. den dazugehörigen Mikrocontrollern nur einigen wenigen Firmen möglich war. Doch mittlerweile sind Chipkarten und Konfigurationsprogramme von verschiedenen Firmen frei zu kaufen. Damit erweitern sich natürlich auch die Möglichkeiten, die einem Angreifer zur Verfügung stehen. Doch unabhängig davon lässt sich mit etwas Aufwand aus einem Kunststoffplättchen, einem Standard-Mikrocontroller im SMD-Gehäuse und etwas Geschick eine funktionsfähige Chipkarte zusammenbauen. Zumindest eine, die sich elektrisch und während der Datenübertragung wie eine echte verhält. Mittlerweile kann diese Art von Chipkarten über die unterschiedlichsten Händler via Internet erworben werden. Neue Möglichkeiten bietet auch die Java-Technologie bei Chipkarten, bei der problemlos eigene Programme erstellt und in eine Dummy-Karte geladen werden können.

Mit einer solchen Dummy-Karte ließe sich zumindest ein Teil der Kommunikation mit dem Terminal protokollieren und später auswerten. Nach mehreren Versuchen ist es dann wahrscheinlich möglich, einen Teil der Kommunikation wie mit einer echten Chipkarte auszuführen.

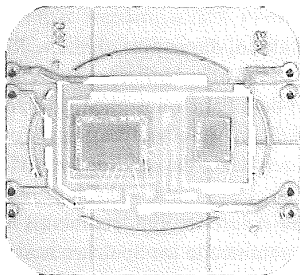


Bild 8.36 Rückseite eines geöffneten Chipkartenmoduls. Der Chip auf der linken Seite ist ein Standard PIC-Mikrocontroller, der mit einem EEPROM-Speicherchip auf der rechten Seite durch Bonddrähte und Leiterbahnen verbunden ist. Diese Art von Chipmodulen wird typischerweise für geklonte Chipkarten oder andere Angriffe auf Chipkartensysteme benutzt.

Ob man daraus auch einen Vorteil erzielen kann, ist zweifelhaft, da alle professionell gestalteten Anwendungen über eine kryptografische Absicherung für wichtige Aktionen verfügen. Solange man den geheimen Schlüssel nicht kennt, ist spätestens bei der Authentisierung das Ende des Angriffs erreicht. Dieser Angriff würde nur zum Erfolg führen, wenn einem der geheime Schlüssel bekannt ist oder die gesamte Anwendung ohne kryptografische Absicherung abläuft. Sollte eine solche Anwendung existieren, darf aber stark bezweifelt werden, ob der durch diesen Angriff erreichbare Vorteil von einer solchen Bedeutung ist, dass der ganze dafür notwendige Aufwand gerechtfertigt ist.

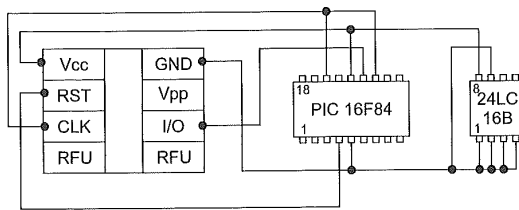


Bild 8.37 Typische Ersatzschaltung für einen Chipkarten-Mikrocontroller, aufgebaut aus diskreten Standardbauelementen (Mikrocontroller PIC 16F84, EEPROM-Speicherchip 24LC16B). Die Bauelemente passen in ein typisches Chipkartenmodul, sodass ohne Untersuchung des Moduls keine Unterscheidung von einem echten Chipkarten-Mikrocontroller möglich ist. Diese Schaltung und Varianten davon finden sich an den einschlägigen Stellen im Internet.

Analyse: Ermittlung des Befehlssatzes einer Chipkarte

Die Befehlsklassen und Kommandos, die eine Chipkarte unterstützt, sind zwar oft nicht veröffentlicht, doch ist es sehr einfach, sie zu ermitteln. Dies ist weniger bei einem Angriff auf die Sicherheit von Interesse, sondern um grundsätzlich den Befehlssatz einer unbekannten Chipkarte zu ermitteln. Es ist vorstellbar, dass auf dieser Grundlage Angriffe geplant werden.

Die Vorgehensweise zur Ermittlung des Befehlsvorrates gestaltet sich entsprechend Bild 8.38. Zuerst bildet man eine Kommando-APDU und sendet diese mit einem frei programmierbaren Terminal zur Chipkarte. Dabei variiert man das Class-Byte von '00' bis 'FF'. Erhält man einen anderen Returncode als „Ungültige Class“, hat man das erste gültige Class-Byte ermittelt. Üblicherweise gibt es 2 bis 3 Befehlsklassen, die man als Grundlage benutzt, um in einer weiteren Runde alle Instruction-Bytes durchzuprobieren. Man sendet dazu so lange Kommando-APDUs mit verändertem Instruction-Byte zur Chipkarte, bis man einen anderen Returncode als „Unbekannte Instruction“ erhält. Mit dieser Methode kann man, die entsprechende Terminal-Software vorausgesetzt, in ein bis zwei Minuten feststellen, welche Kommandos eine Chipkarte unterstützt. Teilweise lässt sich analog dazu auch ein Teil der möglichen Parameter der ermittelten Kommandos herausfinden.

Der dargestellte Algorithmus ließe sich noch erheblich beschleunigen, wenn man nur nach ISO/IEC 7816-4 zulässige Codierungen für das Class Byte und das Instruction Byte als Laufvariablen zulässt. Damit würde der Zahlenraum des Class Bytes durch Berücksichtigung von Secure Messaging und Logischen Kanälen stark reduziert. Analog kann man beim Instruction Byte vorgehen, wenn man nur gerade Zahlen verwendet, da ungerade bekanntermaßen die nicht mehr benötigte Vpp-Kontrolle beinhalten.

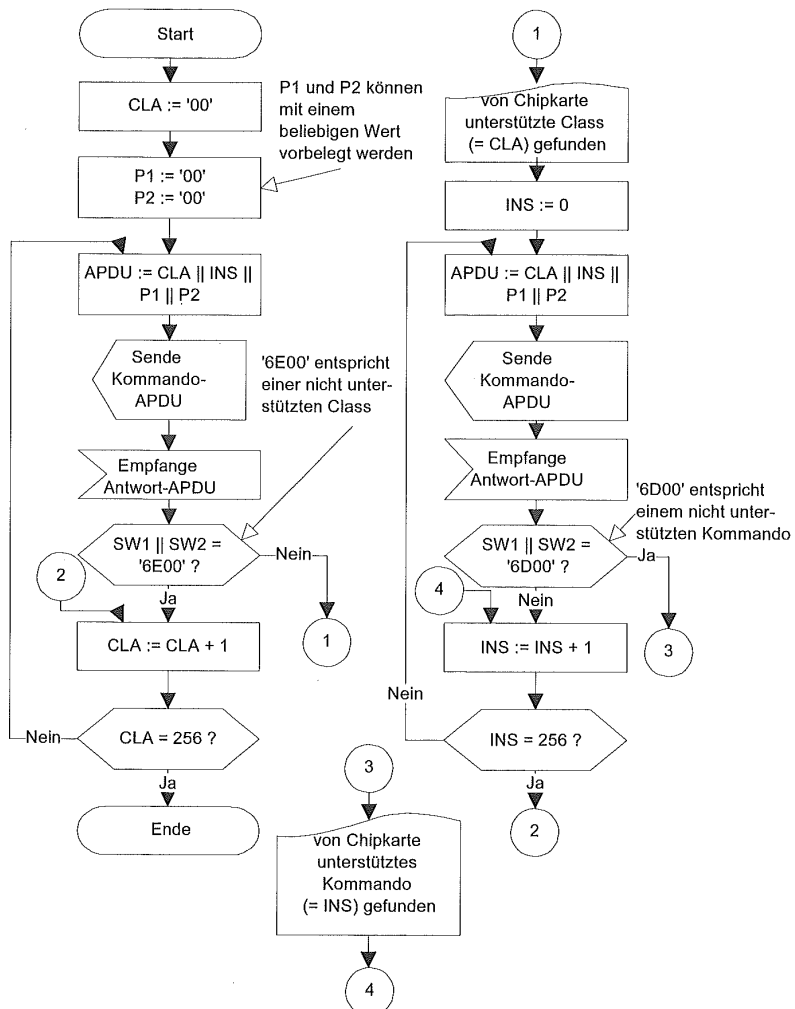


Bild 8.38 Grundsätzliche Vorgehensweise bei der erschöpfenden Suche nach allen von einem Chipkarten-Betriebssystem unterstützten Kommandos. Die Suche funktioniert nur dann lückenlos, wenn der Aufruf von Kommandos nicht durch einen Zustandsautomaten reglementiert ist. Das Prinzip besteht darin, dass alle Class Bytes (CLA) und Instruction Bytes (INS) systematisch ohne Berücksichtigung einer eventuell enthaltenen Codierung (Secure Messaging, Logische Kanäle, Vpp-Kontrolle) durchprobiert werden.

Der Grund, warum dieser einfache Suchalgorithmus für Befehlsklassen, Kommandos und Parameter Erfolg hat, liegt darin, dass praktisch alle Kommandointerpreter der Chipkarten-Betriebssysteme die empfangenen Kommandos vom Class-Byte ausgehend zu den nachfolgenden Bytes hin untersuchen. Beim ersten erkannten ungültigen Wert wird die

Bearbeitung abgebrochen, ein entsprechender Returncode generiert und an das Terminal zurückgesendet.

Die beschriebene Vorgehensweise funktioniert aber nur, wenn die Chipkarte über keinen globalen Zustandsautomaten für die Reihenfolge der Kommandos verfügt. Falls dies der Fall ist, kann man aber mit obigem Verfahren zumindest schrittweise die Reihenfolge der Kommandos herausfinden.

Der Nutzen für einen Angreifer mag nicht besonders groß erscheinen, da üblicherweise der Befehlssatz nicht geheim ist. Doch zumindest können so auf sehr einfache und schnelle Weise alle Kommandos ermittelt werden. Im Übrigen lässt sich diese Methode sehr gut dazu benutzen, um herauszufinden, ob ein Betriebssystemhersteller undokumentierte Kommandos in der Chipkarte untergebracht hat.

Angriff: Abhören der Datenübertragung

Um die Daten während einer Sitzung abzuhören und bei Bedarf manipulieren zu können, verwendet man eine leicht abgeänderte Chipkarte. Auf die I/O-Kontaktfläche wird dazu ein elektrisch isolierter Dummy-Kontakt aufgeklebt. Die ursprüngliche I/O-Schnittstelle ist mit ihm dann elektrisch nicht mehr verbunden. Der auf diese Weise neu geschaffene (Dummy-)Kontakt und der ursprüngliche I/O-Kontakt sind mit einem schnellen Computer verbunden. Dieser kann nun je nach Programmierung bei der Kommunikation zwischen Terminal und Chipkarte beliebige Daten ausschneiden oder einblenden. Ist der Computer hinreichend schnell, wird weder Terminal noch Chipkarte bei der manipulierten Kommunikation einen Unterschied zum regulären Datenaustausch feststellen können.

Man versteht, dass mit dieser Methode der Ablauf einer Sitzung massiv beeinflusst werden kann. Ob nun einem Angreifer daraus ein Vorteil erwächst, hängt vor allem von der Anwendung in der Chipkarte ab. Ein anerkanntes Designkriterium besagt, dass weder durch Abhören, Ausschneiden oder Einblenden von Daten während der Kommunikation die Sicherheit beeinträchtigt sein darf. Wird dieses Kriterium nicht beachtet, kann ein Angreifer auf diese Weise sicherlich einen Vorteil erlangen. Betrugsfälle durch Simulation von Speicherkarten sind bekannt.

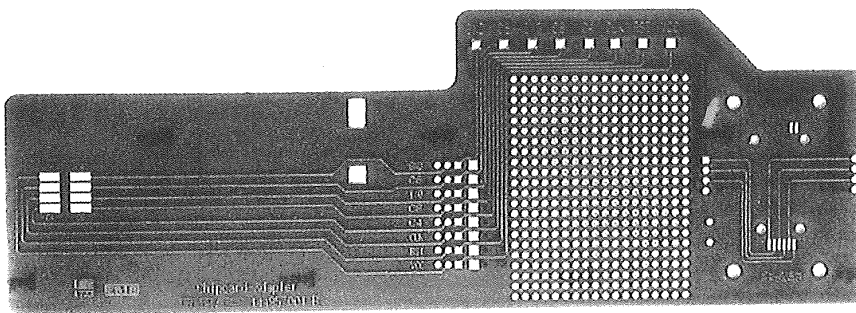


Bild 8.39 Adapter, mit dem eine Chipkarte aus dem Gehäuse eines Kartenterminals zu Messzwecken herausgeführt werden kann. Auf der linken Seite sind die acht Kontaktflächen zu sehen, rechts befindet sich ein Fädelfeld zum Aufbau von Elektronikschaltungen.

Um die Kommunikationsabläufe gegen diesen Angriff abzusichern, haben einige Terminals so genannte Shutter, die alle von der Chipkarte wegführenden Drähte abschneiden. Auch lässt sich hier hervorragend Secure Messaging einsetzen, bei dem Manipulationen während der Datenübertragung zuverlässig erkannt werden.

Da viele Terminals auch nur unter Aufsicht bedient werden dürfen, ist es dort auch nicht so einfach, manipulierte Chipkarten mit Leitungen zu einem mitgeführten Rechner zur Anwendung zu bringen. Rückblickend kann man diesen Angriff zwar als theoretisch sehr interessant und viel versprechend einstufen, doch ist ihm in der Praxis wenig Erfolg beschieden.

Angriff und Abwehr: Abschalten der Stromversorgung

Ein Angriff, der noch vor einigen Jahren bei vielen Chipkarten zum Erfolg geführt hat, ist das Abschalten der Stromversorgung zu einem bestimmten Zeitpunkt während der Ausführung eines Kommandos. Der Hintergrund für diesen Angriff ist die Tatsache, dass bei konventioneller Programmierung alle Schreiboperationen auf EEPROM-Seiten nacheinander ausgeführt werden. Hat nun der Programmierer des Kommandos die Reihenfolge der Schreiboperationen unklug angeordnet, kann das Abschalten der Stromversorgung zum richtigen Zeitpunkt einem Angreifer zum Vorteil gereichen.

An einem stark vereinfachten Beispiel sei dies noch einmal kurz verdeutlicht: Würden bei einem Kommando zum Aufladen einer elektronischen Geldbörse als Allererstes der Börsensaldo erhöht und erst danach die Protokolldateien auf den neuesten Stand gebracht, so hätte ein Angreifer gute Chancen zum kostenlosen Aufladen seiner Chipkarte. Er müsste nur zum richtigen Zeitpunkt die Stromversorgung abschalten oder die Chipkarte millisekundengenau aus dem Kartenterminal herausreißen. Der Dateiinhalt mit dem Börsensaldo wäre dann auf den neuen Wert erhöht, es gäbe dazu jedoch keine Protokolldaten und auch keine Antwort auf das Kommando. Bei simplen elektronischen Börsensystemen wäre in der Vergangenheit ein derartiger Angriff durchaus realistisch gewesen.

Operation	Börsensaldo	Datei mit Börsensaldo (in binärer Schreibweise)
1. aktueller Börsaldo	100 DM	°0110 0100°
2. Abbuchung von 10 DM	90 DM	
3. Löschen des EEPROMs	255 DM	°1111 1111°
4. Schreiben des neuen Börsensaldos	90 DM	°0101 1010°

Bild 8.40 Ablauf beim Schreiben eines neuen Saldos in einer elektronischen Geldbörse. In diesem Beispiel ist angenommen, dass der gelöschte Zustand des EEPROM eins ist. Daraus folgt, dass es halbleitertechnisch notwendig ist, die gesamte EEPROM-Page zu löschen (d.h. auf eins zu setzen), wenn auch nur ein Bit der Page von null auf eins gesetzt werden muss. Würde man in diesem Beispiel den Strom für die Chipkarte exakt nach dem Löschen des EEPROM abschalten (d.h. nach Schritt 3), würde der Börsensaldo auf seinem Maximalwert stehen, und der Angreifer hätte Geld erzeugt. Durch atomare Abläufe lässt sich dies jedoch zuverlässig unterbinden.

Um den genauen Zeitpunkt für den Abbruch zu ermitteln, muss der Angreifer nur mit einem elektronischen Zähler die Takte vom Absenden des Kommandos an zählen und sich dann durch Wiederholungen sukzessive zum richtigen Abschaltzeitpunkt vortasten. Es ist müßig zu erwähnen, dass sich der ganze Vorgang mit einem Computer automatisieren lässt.

Auch wenn dieser Angriff interessant und einfach nachvollziehbar klingt, so gibt es doch in der Praxis einige wirkungsvolle Gegenmaßnahmen. Der einfachste Ansatz ist eine vernünftige Reihenfolge der EEPROM-Schreibbefehle. Die Norm EN 1546 für branchenübergreifende elektronische Geldbörsen ist dazu ein sehr gut geeignetes Anschauungsobjekt, da alle dort beschriebenen Geldbörsen gegen diese Art von Angriff explizit geschützt sind.

Allerdings lässt sich selbst mit einer fehlerfrei ausgetüftelten Reihenfolge der Schreiboperationen kein perfekter Schutz erreichen. Dies sei noch an einem anderen Beispiel verdeutlicht. Bei Aufladen der besagten elektronischen Geldbörse kann es vorkommen, dass das EEPROM vor dem Schreibvorgang gelöscht werden muss. Entspricht nun der gelöschte Zustand des EEPROMs dem Maximalwert der Börse, was im Übrigen meistens der Fall ist, muss nur zum richtigen Zeitpunkt die Stromversorgung abgeschaltet werden, damit die Börse auf ihren Maximalwert aufgeladen ist. Der passende Augenblick zum Abschalten ist dann erreicht, wenn die Löschoperation abgeschlossen ist und die Schreiboperation noch nicht begonnen hat.

Die Betriebssystem-Designer kennen jedoch auch für diesen Angriff eine wirkungsvolle Gegenmaßnahme: die in Abschnitt 5.10 detailliert beschriebenen atomaren Abläufe. Sie besitzen die Eigenschaft, dass sie atomar, also nicht aufteilbar sind. Das bedeutet, dass sie entweder ganz oder gar nicht ausgeführt werden, was als Schutz für den obigen Angriff vollkommen ausreichend ist. Selbst die in der EN 1546 in ihrer Reihenfolge optimal angeordneten EEPROM-Schreiboperationen benötigen an mehreren Stellen atomare Abläufe, damit der oben dargestellte Angriff nicht realisiert werden kann.

Angriff und Abwehr: Stromanalyse bei PIN-Vergleichen

Mit der Kombination von physikalischer Messung eines Parameters und Variation von logischen Werten kann ein technisch sehr interessanter Angriff auf Vergleichsmerkmale, wie z. B. die PIN, unternommen werden. Er betrifft alle Mechanismen, bei denen Daten zur Chipkarte gesendet, dort mit einem gespeicherten Wert verglichen werden, wobei abhängig vom Vergleichsergebnis ein Fehlbedienungszähler erhöht wird.

Das Prinzip ist dabei eine Strommessung der Chipkarte, die beispielsweise über den Spannungsabfall an einem in die Vcc-Leitung eingebrachten Widerstand erfolgt. Sendet man das betreffende Kommando mit den Vergleichsdaten zur Karte, so kann man über die Strommessung vor Erhalt des Returncodes feststellen, ob der Fehlbedienungszähler erhöht wurde oder nicht. Würde nun bei einem positiven Vergleich der Returncode früher ausgesandt, als der Fehlbedienungszähler geschrieben wird, so könnte man auf dieser Grundlage den Vergleichswert ermitteln. Dazu sendet man den Vergleichswert in allen seinen Varianten zur Chipkarte und schaltet diese im Schlechtfall immer vor dem Erhöhen des Fehlbedienungszählers ab. Der Gutfall kann durch den entsprechenden Returncode, der vor der Erhöhung des Fehlbedienungszählers gesendet wird, eindeutig erkannt werden.

Um nun diesen Angriff abzuwehren, gibt es zwei grundsätzliche Methoden: Die einfachste Abwehr besteht darin, dass man den Fehlbedienungszähler grundsätzlich vor jedem

Vergleich erhöht und ihn dann bei Bedarf wieder erniedrigt. Egal zu welchem Zeitpunkt nun ein Angreifer die Spannungsversorgung unterbricht, er kann daraus nie einen Vorteil erzielen, da der Fehlbedienungsähler schon erhöht ist. Die zweite Variante ist etwas aufwendiger, erfüllt aber die gleiche Schutzfunktion. Nach dem Vergleich wird im Schlechtfall der Fehlbedienungsähler erhöht und im Gutfall in eine nicht benutzte EEPROM-Zelle geschrieben. Beide Schreibzugriffe finden zeitgleich statt, sodass ein Angreifer keine Rückschlüsse auf den Vergleich ziehen kann. Er erfährt erst durch den Returncode vom Ergebnis der Vergleichsoperation. Zu diesem Zeitpunkt ist es dann schon zu spät, durch Abschalten der Spannungsversorgung den Schreibzugriff auf den Fehlbedienungsähler zu verhindern.

Angriff und Abwehr: Zeitanalyse bei PIN-Vergleichen

Programmierer achten immer sehr darauf, dass Programme so schnell wie möglich ausgeführt werden. Im Regelfall ist dies auch wichtig. Allerdings lässt sich diese Tatsache der Laufzeitminimierung auch für einen durchaus erfolgversprechenden Angriff nutzen. Wird einer Chipkarte eine PIN zur Prüfung übergeben, so führt die zuständige Vergleichsroutine einen byteweisen Vergleich der übergebenen PIN und der abgespeicherten PIN aus. Ein nicht auf Sicherheit achtender Programmierer wird nun diese Vergleichsroutine so programmieren, dass ein Unterschied beim Vergleich der beiden PIN-Ziffern zum sofortigen Abbruch und Aussprung aus der Vergleichsroutine führt. Damit ergeben sich geringe, doch einem mit geeigneten Instrumentarium (z. B. Speicheroszilloskop) durchaus messbare Laufzeitunterschiede aufgrund des abgebrochenen Vergleichs. Diese können von einem Angreifer benutzt werden, um auf relativ einfache Art und Weise die normalerweise geheime PIN zu ermitteln.

Obiger Angriff war vor einigen Jahren im Bereich der Chipkarten noch erfolgreich. Mittlerweile ist aber diese Art von Angriffen bekannt, und die Vergleichsroutinen sind so ausgelegt, dass prinzipiell immer alle Stellen einer PIN verglichen werden. Damit tritt kein Zeitunterschied zwischen positivem und negativem Vergleichsergebnis auf.

Schutz: Rauschfreier Kryptoalgorithmus

Die Grundlage der Sicherheit einer Chipkarten-Anwendung bilden die geheimen Schlüssel für den kryptografischen Algorithmus. Um bestimmte Zugriffe oder Aktionen auf der Chipkarte ausführen zu dürfen, muss sich das Terminal immer vorher authentisieren, wozu ein geheimer Schlüssel benutzt wird. Es ist verständlich, dass die Authentisierung des Terminals durch die Chipkarte ein interessantes Ziel für einen Angreifer abgibt. Hingegen ist die Authentisierung der Chipkarte durch das Terminal bei einem Angriff auf die Chipkarte uninteressant, da sie von einem (Dummy-)Terminal beliebig manipuliert werden kann.

Das Terminal wird durch die Chipkarte authentisiert, indem es eine Zufallszahl erhält, diese verschlüsselt und dann zurücksendet. Daraufhin führt die Chipkarte die gleiche Verschlüsselung durch und vergleicht das Ergebnis mit dem vom Terminal erhaltenen. Bei Übereinstimmung ist das Terminal authentisiert und erhält einen entsprechenden Returncode. Ist die Authentisierung misslungen, sendet die Chipkarte einen anderen Returncode. Der Ansatzpunkt für einen Angreifer ist nun eine Analyse der Bearbeitungszeit zwischen Kommando und Antwort von der Chipkarte.

Noch in den frühen 90er-Jahren wurden manchmal Kryptoalgorithmen verwendet, die erhebliche Unterschiede in der Ausführungszeit abhängig vom Schlüssel und Klartext hatten. Mit dem so reduzierten Schlüsselraum als Grundlage kann der Angreifer mit einem Brute-force-Angriff nach dem geheimen Schlüssel suchen. Wie lange die Suche dauert, hängt sehr stark vom Rauschen des Algorithmus ab. Je größer aber die Zeitunterschiede sind, desto kleiner ist der Schlüsselraum und desto einfacher und schneller die Schlüssel-suche. Falls die exakte Implementation des entsprechenden Kryptoalgorithmus auf dem Zielrechner bekannt ist, kann dies zusätzlich als Referenz zur Erstellung von Zeittabellen herangezogen werden. Publik gemacht wurde diese Art des Angriffs – eine „Timing Attack“ – 1995 in einer Veröffentlichung von Paul Kocher [Kocher 95], die sich vor allem mit Zeitabhängigkeiten bei RSA und DSS beschäftigte.

Im Prinzip ist eine Timing Analysis für die Sicherheit einer Chipkarte sehr gefährlich. Da er aber schon seit längerer Zeit bekannt ist, benutzen alle heutigen Chipkarten nur noch rauschfreie Kryptoalgorithmen, d. h. die Zeit für Ver- und Entschlüsselung ist unabhängig von den Eingangswerten. Damit wurde diese Art des Angriffs abgeblockt. Allerdings besteht dabei für die Programmierer ein Zielkonflikt, da ein rauschfreier Kryptoalgorithmus meist mehr Programmcode benötigt und immer langsamer ist als die rauschende Variante. Der Grund dafür besteht darin, dass der Algorithmus so beschaffen sein muss, dass die Programmablaufpfade für alle Klartextdaten, Schlüsseltextdaten und Schlüssel immer gleich lang sind. Damit ist der längste notwendige Ablaufpfad die Referenz, der alle anderen entsprechend angepasst werden müssen.

Als zusätzliche Sicherheit verfügen in manchen Anwendungen alle Authentisierungsschlüssel noch über eigene Fehlbedienungszähler, sodass sich nur eine bestimmte Anzahl nicht erfolgreicher Authentisierungen durchführen lässt. Hat der Fehlbedienungszähler seinen Maximalwert erreicht, sperrt sich die Chipkarte gegen alle weiteren Authentisierungsversuche.

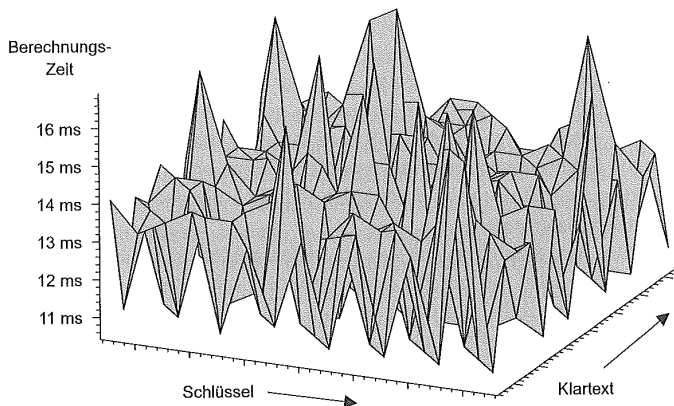


Bild 8.41 Beispiel für den Einfluss von Schlüssel- und Klartextdaten bei einem nicht rauschfreien Verschlüsselungsalgorithmus. Das Bild zeigt einen Ausschnitt aus dem Klar- und Schlüsseltextraum und wurde mit 100 000 Iterationen pro Messwert mit einer alten Implementation des DES-Algorithmus erstellt.

Manipulation: Differenzielle Fehleranalyse (*differenzial fault analysis* – DFA)

Elektronische Geräte lassen sich bekanntermaßen erheblich in ihren Funktionen beeinträchtigen, wenn sie elektromagnetischen Störungen ausgesetzt werden. So kann man beispielsweise die Prozessoren vieler computergesteuerter Kleingeräte ohne weiteres durch ein eingeschaltetes Mobiltelefon zum Absturz bringen. Die Ursache liegt in Speicherzellen, deren Inhalt durch die hochfrequenten Wechselfelder verändert wird.

Im Jahr 1996 veröffentlichten Dan Boneh, Richard DeMillo und Richard Lipton eine Arbeit [Boneh 96], die ein theoretisches Modell beschreibt, wie geheime Schlüssel von asymmetrischen Kryptoalgorithmen durch Einstreuung von Hardwarefehlern berechnet werden können. Da die drei Entdecker damals bei den Bell Communications Research (Bellcore) Laboratories beschäftigt waren, wird dieser Angriff oft auch Bellcore-Angriff (*Bellcore-Attack*) genannt.

Nur zwei Monate später publizierten Eli Biham und Adi Shamir eine Erweiterung des Bellcore-Angriffs mit dem Namen differenzielle Fehleranalyse (*differenzial fault analysis* – DFA) [Biham 96], die nunmehr auch symmetrische Kryptoalgorithmen, wie den DES-Algorithmus, mit einschloss. Damit waren, zumindest in der Theorie, viele Chipkarten-Anwendungen von einer neuen, ernst zu nehmenden Angriffsmethode betroffen.

Das grundlegende Prinzip beider Angriffe ist verhältnismäßig einfach: Im ersten Schritt verschlüsselt man einen beliebigen Klartext mit dem zu brechenden Schlüssel und bewahrt den erhaltenen Schlüsseltext auf. Anschließend wird die Chipkarte während der Abarbeitung des kryptografischen Algorithmus in ihrer Arbeitsweise von außen beispielsweise durch ionisierende oder hochfrequente Strahlung gestört, sodass sich ein einzelnes Schlüsselbit an beliebiger Stelle bei der Berechnung verändert. Das Ergebnis davon ist ein Schlüsseltext, der aufgrund des gekippten Bits falsch verschlüsselt wurde. Dies wird nun mehrmals wiederholt, und die Ergebnisse werden zur Analyse aufbewahrt. Der Rest zur Ermittlung des Schlüssels ist pure Mathematik und in den oben aufgeführten Arbeiten umfassend dargestellt.

Die Stärke des Angriffs liegt vor allem darin, dass es nicht einmal erforderlich ist zu wissen, an welcher Stelle des geheimen Schlüssels ein Bit gekippt wurde. Biham und Shamir geben in ihrer Veröffentlichung an, dass bei einem verfälschten Schlüsselbit bereits 200 Schlüsseltextblöcke genügen, um daraus den geheimen DES-Schlüssel zu berechnen. Bei Verwendung eines echten Triple-DES (168 Bit) anstelle des DES erhöht sich die Zahl der notwendigen Schlüsseltexte nur unwesentlich. Selbst wenn mehr als ein Bit verändert wird, greift dieser Angriff noch. Es erhöht sich lediglich die notwendige Anzahl der falsch verschlüsselten Schlüsseltexte.

So anspruchslos, wie sich diese Art von Angriffen anhört, ist sie in der Praxis dann doch nicht. Es sollten möglichst nur ein Bit oder zumindest sehr wenige Bits verändert werden. Sendet man aber nun pauschal auf den gesamten Mikrocontroller hochfrequente Mikrowellenstrahlung, ändern sich meist so viele Bits, dass der Prozessor in der Regel sofort rettungslos abstürzt. Deshalb versucht man beispielsweise durch vorsätzlich erzeugte Glitches¹² in der Strom- oder Taktversorgung, die CPU zu einer einzigen falschen Berechnung zu veranlassen. Wenn die Filter an den dazugehörigen Eingangsleitungen solch ein Glitch nicht neutralisieren können, kann es zu der beabsichtigten Fehlrechnung des Prozessors kommen.

¹² Glitches sind sehr kurze Spannungseinbrüche oder -erhöhungen

Eine Chipkarte ist allerdings weder dem Bellcore-Angriff noch einer DFA schutzlos ausgeliefert, wenn zuvor entsprechende Vorsorge praktiziert wurde. Die einfachste Abwehr ist, den Kryptoalgorithmus in der Chipkarte einfach zweimal zu berechnen und die beiden Ergebnisse zu vergleichen. Sind die Resultate identisch, dann wurde nicht versucht, von außen irgendwelche Bits zu kippen. Dabei geht man davon aus, dass eine bewusste Fehlereinstreuung niemals die gleichen Bits in der Chipkarte verändern kann. Dies ist wirklichkeitsnah, denn sollte jemals die gezielte Änderung von bestimmten Bits in einem Chipkarten-Prozessor möglich sein, gibt es viel einfachere und schneller durchzuführende Angriffe als eine DFA.

Der große Nachteil einer Doppelrechnung ist der zusätzliche Zeitbedarf, welcher Probleme bereiten kann. Dies betrifft vor allem Angriffe auf zeitaufwendige asymmetrische Kryptoverfahren wie RSA oder DSS. Eine weitere wirksame Abwehrmaßnahme gegen differenzielle Fehleranalysen kann dadurch bewerkstelligt werden, indem man immer nur unterschiedliche Klartexte verschlüsselt. Die einfachste Lösung ist eine dem zu verschlüsselnden Klartext vorangestellte Zufallszahl. Damit verschlüsselt der Kryptoalgorithmus immer verschiedene Daten, und eine DFA ist nicht mehr möglich.

Die Quintessenz aus Bellcore-Angriff und differenzieller Fehleranalyse lässt sich auf die Formel bringen, wonach es sich um durchaus gefährliche Angriffe handelt, die bei unzureichend ausgestatteten Chipkarten zum Erfolg führen können. Doch wurden innerhalb kurzer Zeit nach Bekanntwerden der beiden Angriffsmethoden alle Chipkarten-Betriebssysteme und Anwendungen diesbezüglich abgesichert, sodass sowohl Bellcore-Angriff als auch DFA heute keine ernsthafte Gefährdung mehr darstellen.

Angriff und Schutz: Störung des Prozessors

Ähnlich der Benutzung der differenziellen Fehleranalyse DFA beim Angriff auf die geheimen Schlüssel von Kryptoalgorithmen kann man durch Störung des Prozessors versuchen, Abläufe im Programmcode zu beeinflussen. Der sowohl bei den Herstellern von Chipkarten wie bei Chipkarten-Mikrocontrollern und einigen Systemhäusern seit ca. 1998 als Lichtangriff (*light attack*) bekannte Angriff wurde Mitte 2002 von Sergei Skorobogatov und Ross Anderson [Skorobogatov 02] als *Optical Fault Induction Attacks* veröffentlicht. Die Publikation beschreibt eine Anordnung, bei der ein handelsübliches Blitzgerät auf den Kameraadapter eines konventionellen Lichtmikroskops angeflanscht ist. Anschließend wird ein eng begrenzter Bereich des RAMs eines Standard-Mikrocontrollers (PIC16F84) angeblitzt. Mit dieser Anordnung ist es möglich, selektiv bestimmte Bits im RAM dieses – nicht gegen diese Art von Angriffen gehärteten – Mikrocontrollers in den Zustand 0 oder 1 zu setzen.

Um den Prozessor zu stören, können beispielsweise Glitches auf den Versorgungsleitungen, Lichtblitze auf den Chip oder Teile des Chips oder auch Hochfrequenz verwendet werden [Lamla 00]. Wird dann zum richtigen Zeitpunkt im Programmablauf die Störung ausgelöst, so lässt sich damit beispielsweise eine Abfrage gezielt beeinflussen. Bild 8.42 zeigt dazu ein einfaches Beispiel. Die dargestellte Programmfunktion hat die Aufgabe, den Inhalt eines Sendepuffers zu übertragen, dessen Grenzen von einer Anfangsadresse und einer Endadresse festgelegt sind. Schafft es der Angreifer, gezielt die Abfrage auf das Ende des Sendepuffers zu stören, werden auch die Daten an das Terminal übertragen, die im Anschluss an den Sendepuffer folgen. Befände sich nun in diesem Speicherbereich der Arbeitsspeicher für geheime Schlüssel eines Kryptoalgorithmus, ließen sich die Schlüssel mit dieser Methode unberechtigtweise auslesen.

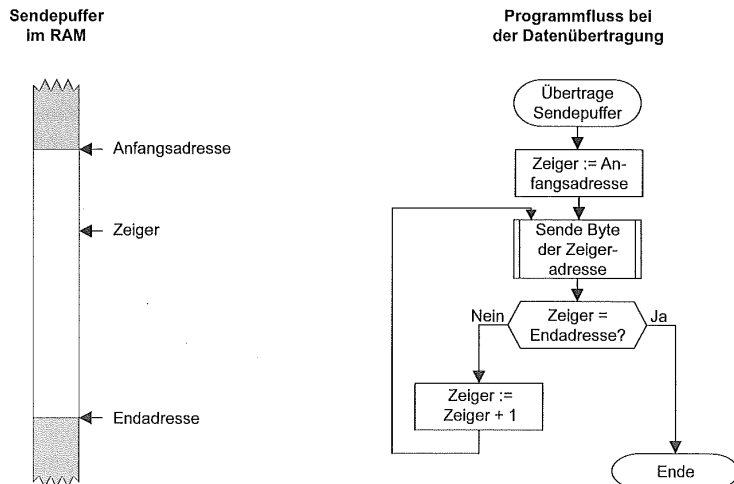


Bild 8.42 Beispiel für einen nicht robusten Programmablauf zum Übertragen eines Sendepuffers, der durch Störung des Prozessors mit Erfolg angegriffen werden kann.

Die Abwehr dieses Angriffs gestaltet sich vielschichtig. Wichtig ist, dass der Chipkarten-Mikrocontroller über entsprechende Sensoren verfügt, um alle Störungsversuche des Prozessors zu erkennen. Dies können auf Glitches detektierende Spannungssensoren und eine große Anzahl entsprechender Lichtsensoren auf dem Chip sein. Damit es nicht möglich ist, einige wenige Lichtdetektoren mit schwarzer Tusche zu maskieren, ist es sinnvoll, eine größere Anzahl von Lichtdetektoren vorzusehen. So können bereits viele Angriffe vom Ansatz her verhindert werden. Eine opake Chipvergussmasse würde nur eingeschränkten Schutz bieten, da diese relativ einfach auf chemischen Wege entfernt werden kann.

Die zweite Schutzebene muss in der Software realisiert werden. Der im Beispiel dargestellte Programmcode kann durch Ersatz der „=“-Abfrage durch eine „<=-“-Abfrage deutlich robuster gemacht werden. Weiterhin kann als Gegenmaßnahme die Anfrage doppelt ausgeführt werden, wobei der zeitliche Abstand der beiden Abfragen eine zufällige Länge haben sollte. Damit müsste der Angreifer zwei Lichtblitze zur Manipulation der Abfrage einsetzen und stünde zudem noch vor dem Hindernis, dass er den Zeitpunkt für den zweiten Lichtblitz nicht exakt vorhersagen kann.

Zusätzlich sollten alle vertraulichen Daten im RAM unmittelbar nach ihrer Benutzung wieder gelöscht oder temporär verschlüsselt werden. Um die Auswirkungen dieses Angriffs weiter zu reduzieren, ist es ebenfalls sinnvoll, alle Geheimnisse (z. B. PIN, Schlüssel) im EEPROM zu verschlüsseln. Sollte es nämlich ein Angreifer schaffen, Teile des EEPROMs durch Manipulation von Abfragen auszulesen, so würde er als Ergebnis nur verschlüsselte Daten erhalten, die ihm nichts nützen. Steht eine MMU zur Verfügung, kann diese zusätzlich auf die Überwachung der Einhaltung von bestimmten Grenzen beim Senden von Daten konfiguriert werden. Außerdem können moderne Prozessoren illegale Maschinenbefehle und ungültige Adressen erkennen und darauf entsprechend reagieren. Dieses Abwehrszenario zeigt sehr gut, dass bei entsprechendem Zusammenspiel von Hard- und Software-schutzmaßnahmen ein durchaus ernst zu nehmender Angriff abgeblockt werden kann.

Schutzkomponente: Chipkarten-Betriebssystem

Die Schutzmechanismen der Hardware sind die Grundlage für die Schutzmechanismen der Betriebssystemsoftware. Dabei darf keine Lücke übersehen werden, da die drei Teile der Schutzmechanismen (Hardware, Betriebssystem, Anwendung) mit einem logischen „UND“ verknüpft sind. Dies ist vergleichbar einer Kette, bei der das schwächste Glied die Zugfestigkeit bestimmt. Fällt bei einer Chipkarte ein Mechanismus aus, bricht die gesamte Sicherheit zusammen. Gerade das Betriebssystem ist wiederum auch die Basis für die eigentliche Anwendung, deren Informationen und Abläufe geschützt werden müssen.

Die folgenden Abschnitte behandeln insbesondere Schutzmechanismen gegen typische Angriffe und weniger allgemeine Sicherheitsfunktionen eines Chipkarten-Betriebssystems. Allerdings sind auch diese zu einem großen Teil mitverantwortlich für Betriebssicherheit und Schutz gegen Angriffe, weshalb hier ausdrücklich auf die entsprechenden Abschnitte in Kapitel 5 hingewiesen sei.

Schutz: Hard- und Softwaretest nach Reset

Beim Initialisieren des Betriebssystems müssen grundsätzlich die wichtigsten Teile der Hardware auf korrekte Funktionsfähigkeit geprüft werden. So ist eine Prüfung des RAMs unabdingbar, da dort während der Laufzeit alle Zugriffsbedingungen gespeichert sind und ein Ausfall eines bestimmten Bits zum Zusammenbrechen der gesamten Sicherheit führen kann. Genauso unverzichtbar ist die Berechnung und der Vergleich von Prüfsummen über die wichtigsten Teile des ROM und EEPROM. Ein Test der CPU ist zumindest implizit durch das Senden des ATRs gegeben, da dazu der überwiegende Teil aller möglichen Maschinenbefehle fehlerfrei ausgeführt werden muss. Explizite Tests der CPU oder einer eventuell vorhandenen NPU können sich jedoch in der Regel nur auf Stichproben beschränken, da eine vollständige Prüfung auf einwandfreie Funktion viel zu zeit- und codeaufwendig wäre.

Falls das Betriebssystem einen Hardware- oder Prüfsummenfehler entdeckt, existieren zwei Vorgehensweisen: Die Software kann sofort in eine Endlosschleife springen – die Chipkarte sendet dann keinen ATR mehr aus, und die folgenden Kommandos können ebenfalls nicht mehr empfangen werden. Der große Nachteil besteht hier darin, dass die Ursache dieses Verhaltens von außen nicht erkennbar ist. Man weiß beispielsweise nicht mehr, ob nun ein Chipbruch, der Riss eines Bonddrahtes oder ein Prüfsummenfehler im EEPROM vorliegt. Günstiger ist es deshalb, wenn die Chipkarte versucht, einen speziellen ATR zu senden, und sich dann über eine Endlosschleife abschaltet. Durch diesen Fehler-ATR ist für die äußere Welt zumindest noch im Ansatz erkennbar, was innerhalb der Chipkarte passiert ist. Allerdings darf keinesfalls übersehen werden, dass bereits das Aus-senden eines Fehler-ATRs eine weitgehend funktionsfähige CPU, einige Bytes im RAM und mehrere hundert Bytes Programmcode im ROM erfordert.

Schutz: Schichtentrennung des Betriebssystems

Eine Schichtentrennung mit klar definierten Übergangsparametern zwischen den einzelnen Schichten ist ein Merkmal für ein stabiles und robustes Chipkarten-Betriebssystem. Auswirkungen von möglicherweise im Betriebssystem enthaltenen Programmier- oder Konzeptionsfehlern werden durch eine saubere Schichtentrennung innerhalb des Betriebssystems minimiert. Dies bedeutet zwar nicht, dass solche Fehler dann nicht auftreten, doch die Auswirkungen sind nicht so groß wie bei einem in sehr komprimiertem und zusammenge-

fasstem Code programmierten Betriebssystem, da sich Fehler in einer Schicht aufgrund der Trennung nur schwer in eine andere hinein fortsetzen können.

Schutz: Kontrolle der Datenübertragung

Ein weiteres sehr wichtiges Sicherheitselement, die Kontrolle der Datenübertragung, schützt den Speicher vor unerlaubten Zugriffen. Die gesamte Kommunikation zur und von der Chipkarte läuft über die vom Betriebssystem kontrollierte I/O-Schnittstelle. Es sind keine anderen Zugriffe möglich. Dies stellt den wirksamsten Speicherschutz in der Chipkarte dar, denn auf diese Weise ist sichergestellt, dass das Betriebssystem in jedem Fall die Kontrolle über Zugriffe auf Speicherbereiche behält.

Das Übertragungsprotokoll, das vom Transportmanager gesteuert wird, muss alle möglichen Fälschungen abfangen. Es darf keine Möglichkeit bestehen, durch Manipulationen der Übertragungsblöcke die Datenübertragung so zu beeinflussen, dass unberechtigt falsche Daten aus dem Speicher zum Terminal gesendet werden.

Schutz: Prüfsummen über wichtige Speicherinhalte

Die Dateiorganisation und dabei insbesondere die Header, also die Dateideskriptoren, sollte mit Prüfsummen abgesichert sein. Ändert sich ein Speicherinhalt unbeabsichtigt, kann dies wenigstens durch das Betriebssystem detektiert werden. In Hinblick darauf, dass sich die jeweiligen objektorientierten Zugriffsbedingungen in diesem Teil der Datei befinden, ist diese Forderung sehr wichtig.

Alle Speicherbereiche im EEPROM, die für das Betriebssystem der Chipkarte von existenzieller Bedeutung sind, müssen durch Prüfsummen (EDC) abgesichert sein. Bei Zugriffen oder Aufrufen in diesen Bereichen muss vorab die Konsistenz des Inhalts mittels der Prüfsumme festgestellt werden, damit durch Speicherfehler im EEPROM die Stabilität des Betriebssystems nicht gefährdet ist.

Schutz: Kapselung von Anwendungen

Einige Betriebssysteme kapseln die einzelnen DFs, in denen sich die Anwendungen mit ihren Dateien befinden, voneinander ab. Diese Konzepte basieren aber nur auf einem reinen Softwareschutz, ohne Hardware-Unterstützung des Chips. Dadurch ist der Schutzeffekt nicht so groß, wie er es sein könnte. Doch ist selbst die Software-Lösung der Anwendungskapselung im Fehlerfall von großem Vorteil, da es auf diese Weise dem Dateimanagement unmöglich gemacht wird, die Grenzen eines DF ohne vorherige explizite Selektion zu überschreiten. Damit werden die Auswirkungen eines Speicherfehlers an einer Datei zumindest auf das jeweilige DF begrenzt.

Bei einer vorhandenen Hardware-Unterstützung des Betriebssystems durch eine MMU (*memory management unit*) lassen sich unterschiedliche Anwendungen vollständig voneinander abkapseln, sodass selbst durch manipulierte Software innerhalb einer Anwendung kein unerlaubter Zugriff auf Speicherbereiche anderer Anwendungen mehr möglich ist.

Schutz: Verschleierung von Betriebssystemoperationen

Das Schreiben von Daten in das EEPROM erfordert das Anschalten der auf dem Chip befindlichen Ladungspumpe. Dadurch steigt der Stromverbrauch der Chipkarte, was bei manchen Chipkarten-Mikrocontrollern anhand einer entsprechenden Messanordnung erkennbar

ist. Dies bedeutet, dass man bei der Gestaltung des Betriebssystems darauf achten muss, dass Schreibzugriffe auf das EEPROM außerhalb der Chipkarte feststellbar sind. Die Software in der Chipkarte muss verhindern, dass ein Angreifer daraus einen Vorteil erzielen kann.

Deshalb ist es sehr wichtig, dass anhand von Strommessungen keine verwertbaren Aussagen über interne Abläufe und Entscheidungen im Maschinenprogramm zu machen sind. Es wäre beispielsweise fatal, wenn über eine Strommessung vor dem Abschluss eines Kommandos und Erhalt des entsprechenden Returncodes eine verlässliche Aussage über einen PIN-Vergleich möglich wäre; dies ließe sich nämlich sehr gut zu einer Analyse der PIN ausnutzen.

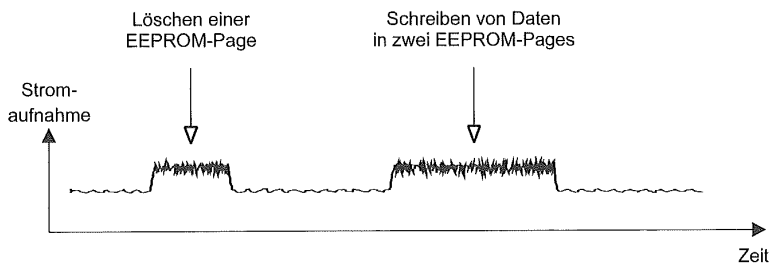


Bild 8.43 Annäherungsweise Darstellung des Stromverbrauchs eines Chipkarten-Mikrocontrollers beim Anschalten der Ladungspumpe.

Schutz: Objektorientierte Zugriffsbedingungen

Frühere Chipkarten-Anwendungen bauten immer auf einem zentral verwalteten Zugriffsmechanismus auf. Ein Nachteil dieser zentralistisch aufgebauten Zugriffsautomaten ist die Tatsache, dass sich Software- oder Speicherfehler auf die gesamte Sicherheit der Chipkarte auswirken. Moderne objektorientierte Dateiverwaltungssysteme mit den Zugriffsrechten an den einzelnen Dateien haben den Vorteil, dass bei Speicherfehlern nur eine einzige Datei davon betroffen ist und die Sicherheit der restlichen Dateien weiterhin gewährleistet bleibt. Dies ist eigentlich eine grundlegende Eigenschaft aller verteilten Systeme. Sie sind in der Programmierung etwas aufwendiger, doch in der Sicherheit durch ihre Autarkie wesentlich stärker gegenüber Angriffen oder Fehlern.

Schutz: Deaktivierung der Chipkarte

Das Betriebssystem muss eine Möglichkeit vorsehen, die Chipkarte komplett zu deaktivieren. Dies ist für die letzte Phase des Lebenszyklus sehr wichtig. Durch das Sammeln von abgelaufenen, aber voll funktionstüchtigen Chipkarten lassen sich mit statistischen Methoden sehr genaue Analysen der Software im Chip durchführen. Um genau dies zu verhindern, müssen im Betriebssystem Mechanismen vorhanden sein, um das Betriebssystem mit allen seinen Programmteilen komplett zu deaktivieren und auf diese Weise elektrische oder Laufzeitanalysen unmöglich zu machen.

Angriff und Abwehr: Zufallszahlengenerator

Die von der Chipkarte erzeugten Zufallszahlen benutzt man bei der Authentisierung, um eine Sitzung zu individualisieren, d. h. um sie einzigartig und anders als alle vorangegangenen und auch die nachfolgenden zu machen. Der Sinn dieses Vorgehens liegt darin, dass

man ein erfolgreiches Wiedereinspielen von früher abgehorchten Sitzungen unmöglich macht. Eine Variante des Angriffs bestünde nun darin, sich von der Chipkarte so viele Zufallszahlen erzeugen zu lassen, dass man sie vorhersagen kann. Eine andere Möglichkeit wäre, so lange Zufallszahlen von der Chipkarte anzufordern, bis der EEPROM-Speicher des Zufallszahlengenerators nicht mehr funktioniert und dann immer die gleichen Zahlen generiert werden.

Alle diese Angriffe könnten im Falle des Erfolges eine Authentisierung des Terminals durch die Chipkarte umgehen. Sie funktionieren ausnahmslos nur bei den Chipkarten der ersten Generationen. Bei modernen Betriebssystemen würden sie fehlschlagen. Die Periode der erzeugten Zufallszahlen ist mittlerweile so groß, dass die Zufallszahlen innerhalb der Lebensdauer der Chipkarte nie zweimal auftauchen. Auch nutzt es nichts mehr, so viele Zufallszahlen zu erzeugen, bis Schwierigkeiten mit dem EEPROM eintreten. Sollte dies der Fall sein, wird einfach die Generierung der Zufallszahlen blockiert und jede weitere Authentisierung verhindert.

Ein hochwertiger Zufallszahlengenerator muss noch weitere Anforderungen – wie nicht vorhersagbare Zufallszahlen und eine große Periode bei den erzeugten Zufallszahlen – erfüllen. Zudem müssen alle Chipkarten einer Anwendung unterschiedliche Zufallszahlen generieren, was äußerst banal klingt, in der Vergangenheit aber immer wieder zu Problemen geführt hat. Realisiert wird das unterschiedliche Verhalten durch einen bei der Initialisierung oder Personalisierung in die Chipkarte eingebrachten Startwert des Pseudozufallszahlengenerators. Dieser Initialwert wird in Anlehnung an landwirtschaftlichen Ackerbau oft auch *seed number* genannt. Aufbau und Kriterien für Pseudozufallszahlengeneratoren sowie Messmethoden für die Güte von Zufallszahlen sind unter „4.10 Zufallszahlen“ ausführlich erläutert.

Schutzkomponenten der Chipkarten-Anwendung

Die Schutzmechanismen der Anwendung setzen auf den entsprechenden Mechanismen der Hardware und des Betriebssystems auf. Die Anwendung ist darauf angewiesen, dass die beiden unteren Schichten ihren Schutzaufgaben vollständig nachkommen, denn ein Fehler in der Hardware oder im Betriebssystem kann durch sie nicht mehr abgefangen werden. Kann man beispielsweise das EEPROM mit einem Analyseverfahren auslesen, nützen die aufwendigsten und sichersten Verschlüsselungsverfahren nichts, da die Schlüssel vom Angreifer direkt dem EEPROM entnommen werden können. Eine Anwendung muss jedoch so aufgebaut sein, dass im Falle eines erfolgreichen Angriffs einer Karte nicht das gesamte System kompromittiert wird.

Schutz: Einfache Mechanismen

Um sich gegen Angriffe wirkungsvoll zu schützen, sollte man alle Mechanismen einer Anwendung möglichst einfach gestalten und sich allzeit an das generell gültige Prinzip „keep it as simple as possible“ halten. Denn dies erleichtert schon am Anfang die Realisierung und anschließend die Prüfung auf korrekte Funktionsfähigkeit von Mechanismen und Wirksamkeit von Schutzmaßnahmen. Es ist äußerst gefährlich anzunehmen, dass man gegen alle möglichen Attacks gewappnet ist, wenn man es nur genügend kompliziert macht. In der Regel wird es genau andersherum sein. Komplizierte Abläufe und Mechanismen führen meist dazu, dass diverse Dinge vergessen und übersehen werden und ein Angreifer es dann umso leichter hat.

Grundsätzlich sollten immer die vom Chipkarten-Betriebssystem zur Verfügung gestellten Schutzmechanismen für die Anwendung benutzt werden. Diese sind geprüft, zuverlässig und wehren bereits auf niedriger Softwareebene Angriffe ab. Dies soll nicht heißen, dass eine Anwendung keine eigenen Schutzmechanismen mehr haben muss, doch sollten die bereits im Betriebssystem vorhandenen in jedem Falle genutzt werden.

Schutz: Konservative Zugriffsrechte

Neben dem Prinzip „keep it simple“ gibt es noch eine zweite allgemein gültige Regel. Sie besagt, dass Zugriffsrechte auf Dateien oder Kommandos einer Chipkarte so konservativ wie möglich zu vergeben sind. Generell sollte der Zugriff verboten sein und nur die unbedingt notwendigen erlaubt werden.

Die Vorteile liegen darin, dass es auf diese Weise weniger wahrscheinlich ist, unbeabsichtigt Zugriffe auf wichtige Daten und Kommandos zu vergeben und dass ein Angreifer für jedes Stück an benötigter Information Aufwand betreiben muss. Damit kann man die Angriffsattraktivität erheblich reduzieren, da man den Aufwand hochtreibt, der für den Angriff notwendig ist.

Schutz: Zustandsautomaten für Kommandosequenzen

Ein Angriff auf eine Chipkartenanwendung wird erheblich erschwert, wenn nicht jedes Kommando zu jedem Zeitpunkt und in unbegrenzter Zahl ausgeführt werden darf. Realisiert wird dies durch einen Zustandsautomaten, mit dem die erlaubte Reihenfolge von Kommandos festgelegt werden kann. Legt man dabei beispielsweise als Erstes eine gegenseitige Authentisierung von Terminal und Chipkarte fest, hat der Angreifer zunächst diesen Schutzwall zu überwinden, bevor er weitere Kommandos ausführen kann.

Schutz: Doppelte Zugriffssicherung

Dem Angreifer wird seine Aufgabe erheblich erschwert, wenn die Dateien der Chipkarte nicht nur durch die am Objekt angebrachten Zugriffsbedingungen geschützt sind, sondern zusätzlich die Kommandos und dazugehörigen Parameter durch einen Zustandsautomaten fest vorgegeben sind. Denn dann kann er nicht mehr durch bloßes Durchprobieren von Kommandos oder Kombination von Kommandos systemspezifische Eigenheiten aufdecken und daraus eventuell einen Nutzen ziehen. Mit von einem Zustandsautomaten überwachten Kommandosequenzen können nur noch die in der Anwendung definierten Kommandos in der Chipkarte ausgeführt werden, alles andere wird schon vor der Ausführung durch den Zustandsautomaten abgeblockt. Dies verringert in erheblichen Umfang die Möglichkeiten, die einem Angreifer durch Kommandomanipulation offen stehen.

Schutz: Verschiedene Prüfebenen

Bei Banknoten ist es seit Jahrzehnten Stand der Technik, mehrere Prüfebenen zu unterstützen. Damit sind Sicherheitsmerkmale gemeint, die von unterschiedlichen Personen- oder Maschinengruppen unabhängig voneinander geprüft werden können. So sind viele der optischen Merkmale, wie Sicherheitsfaden und Wasserzeichen, von jedermann auf der Straße prüfbar. Zur Prüfung der nächsten Ebene benötigt man schon eine UV-Lampe, um die fluoreszierenden Farbpigmente im Papier zu erkennen. Die darüber befindliche Prüfebene enthält Merkmale, welche von den Automaten zur Feststellung der Echtheit genutzt wer-

den. Ein typisches Beispiel sind die Infraroteigenschaften der Banknote. Den Zentralbanken stehen dann wiederum eigene und unabhängige Merkmale für ihre Prüfungen zur Verfügung.

Das ganze System lässt sich problemlos auf Chipkarten übertragen und führt dann in logischer Konsequenz dazu, dass nicht jede Instanz alles prüfen kann. So ist es vorstellbar, dass in einem elektronischen Geldbörsensystem ein Händlerterminal nicht alle Schlüssel zur Prüfung der Signaturen besitzt, sondern nur einen Teil davon. Das System wird dadurch kryptografisch nicht geschwächt, erhält aber den Vorteil, dass ein Angreifer nicht mehr das gesamte System diskreditiert, wenn er die Hauptschlüssel eines Händlerterminals in Erfahrung bringt. Einzig und allein der Systembetreiber kennt alle Schlüssel, die für einen vollständigen Transaktionsdatensatz notwendig sind, und kann dann auf jeden Fall einen Angriff anhand der gefälschten Signaturen erkennen und entsprechende Gegenmaßnahmen einleiten.

Schutz: Sicherheitsmerkmale

Zusätzliche Sicherheit beim Betrieb einer Chipkarte können Merkmale auf dem Mikrocontroller bieten. Diese bestehen aus extra aufgebrachten Funktionseinheiten, die parallel zur Software im Chip vom Terminal geprüft werden können. Dabei verwendet man sowohl analoge als auch digitale Baugruppen. Die Sicherheit dieser Merkmale ist auf Verschleierung begründet und für jede Anwendung variabel, sodass man anwendungsspezifische Chips erhält.

Schutz: Gesicherte Datenübertragung

Die Datenübertragung in unsicherer Umgebung beherbergt einige Risiken. Es können durch nicht allzu aufwendige technische Manipulationen an der Schnittstelle zwischen Terminal und Karte fast beliebige Daten während einer Sitzung hinzugefügt oder aus dem normalen Ablauf herausgeschnitten werden. Geschieht dies während der Übertragung von sicherheitssensiblen Daten, kann ein Angreifer unter Umständen daraus einen Vorteil erzielen. Um diese nicht allzu aufwendigen oder schwierigen Angriffe zu unterbinden, kann man die Verfahren von Secure Messaging einsetzen. Die komplette Verschlüsselung aller übertragenen Daten sollte aber dennoch vermieden und nur zur Übertragung von z. B. geheimen Schlüsseln verwendet werden. Ein Argument gegen die komplette Verschlüsselung ist datenschutzrechtlicher Natur. Fast alle Daten, die in den Speicher der Chipkarte geschrieben werden, sind öffentlich. Verschlüsselt man nun diese Daten, kann niemand mehr nachprüfen, was eigentlich in die Chipkarte geschrieben bzw. aus ihr gelesen wurde. Um hier im Ansatz berechtigten Bedenken aus dem Wege zu gehen, sollten die Daten so weit wie möglich nicht während der Übertragung verschlüsselt sein.

Schutz: Fehlerbehebungsfunktionen

Kommt es zu einem undefinierten Abbruch einer Sitzung oder zu prinzipiellen Unklarheiten über eine vergangene Sitzung, ist es von großem Vorteil, wenn in der Chipkarte anwendungsspezifische Protokolldateien vorhanden sind. Diese werden während der Sitzung vom Betriebssystem mit den jeweils aktuellen Zuständen der Anwendung und eventuell erhaltenen Signaturen oder anderen Daten des Terminals auf dem neuesten Stand gehalten. Untergebracht sind diese Daten in einer zyklischen Datei, wo sie jeweils auf den ältesten

Record geschrieben werden, sodass dessen Inhalt damit verloren geht. Hat diese Datei z. B. 20 Records, können Informationen über die letzten 20 Sitzungen für Analysen des Sitzungsablaufs gespeichert werden. Damit lassen sich viele Unklarheiten beseitigen und strittige Transaktionen und Abläufe unzweifelhaft klären.

Ein weiteres Argument für detaillierte Protokolldateien in der Chipkarte sind die dann möglichen Fehlerbehebungsfunktionen (*error recovery*). Damit ist es möglich, bei einem undefinierten Abbruch automatisch den alten Zustand der Karte wiederherzustellen (*roll back*), ohne dass Analysen des genauen Ablaufs und Hergangs, eventuell sogar mit Personenbeteiligung, notwendig sind.

Schutz: Authentisierung

Die von der Magnetstreifenkarte bekannte einseitige Authentisierung bedeutet letztendlich nur, dass das Terminal feststellt, ob die Karte echt ist. Die Magnetstreifenkarte hat aufgrund ihrer passiven Natur keine Möglichkeit, die Echtheit des Terminals zu überprüfen. Die Einführung der Chipkarte hat dies grundlegend geändert. Nunmehr kann auch die Karte prüfen, ob sie in einem echten Terminal steckt oder Verbindung mit einem echten Hintergrundsystem hat. Dies hat weitgehende Auswirkungen auf die Sicherheit, denn die Chipkarte kann dadurch auch aktiv gegen unerlaubte Zugriffsversuche vorgehen.

Die Möglichkeiten, die mit dieser gegenseitigen Authentisierung einhergehen, sind zahlreich und werden meistens nicht einmal annähernd ausgeschöpft. Zumindest sollte eine Chipkarte sich gegen alle weiteren Zugriffsversuche, egal in welcher Form, sperren, solange sich das Terminal nicht korrekt authentisieren lässt. Dann ist es auch nicht mehr möglich, im stillen Kämmerchen Analysen am Chipkarten-Betriebssystem vorzunehmen, und sei es nur, um alle vorhandenen Kommandos zu ermitteln.

Schutz: Online-Verhalten

Terminals mit integrierten Sicherheitsmodulen können völlig autark Anwendungen mit Chipkarten betreiben. Natürlich sind periodische Up- und Downloads zu einem Hintergrundsystem notwendig, doch das ist normalerweise sehr selten der Fall. Bei größeren Anwendungen mit einer großen Stückzahl von Chipkarten im Feld muss jedoch für die Terminals die Möglichkeit bestehen, bei Bedarf sehr schnell eine Verbindung zum Hintergrundsystem herzustellen, sodass dieses eine direkte Ende-zu-Ende-Kommunikation mit der Chipkarte durchführen kann. Dies ist umso wichtiger, je größer das System ist und je größer der Vorteil ist, den ein Angreifer aus einem Betrug erzielen kann. Denn bei einer direkten Verbindung zwischen Chipkarte und Hintergrundsystem kann dieses auf seine aktuellen Datenbanken zugreifen und gegebenenfalls die Chipkarte sperren. Zudem sind die Schlüssel in einem Hintergrundsystem wesentlich sicherer gespeichert als in den vielen im Feld befindlichen Terminals, auch wenn diese über ein Sicherheitsmodul verfügen. Zusätzlich kann das Hintergrundsystem mit den Kartendaten, die es bei den sporadischen Ende-zu-Ende-Verbindungen erhält, sehr gute statistische Auswertungen über diverse Dateninhalte in der Chipkarte anfertigen.

Alle diese Argumente sind natürlich bei elektronischen Geldbörsen auf Chipkartenbasis von besonderem Interesse. Der Zwang, „online zu gehen“, kann durch eine Zufallsvariable und Zeitlimits, die in der Chipkarte gespeichert sind, ausgelöst werden. Genauso wirksam sind Zähler in der Chipkarte, die bei einer bestimmten Anzahl von Offline-Transaktionen

oder einem bestimmten Offline-Transaktionswert zwingend eine Online-Verbindung mit gegenseitiger Authentisierung verlangen. Das Hintergrundsystem kann diese Zähler anschließend wieder zurücksetzen oder die entsprechenden Parameter des Online-Verhaltens neu festlegen.

Schutz: Sperrlisten

Es ist nie völlig auszuschließen, dass es in einem System irgendwann zu Fälschungen von Chipkarten kommt, selbst wenn diese noch so gut gegen alle Angriffe gesichert sind. Auch zum Schutz der Benutzer muss ein Chipkartensystem wirkungsvolle Mechanismen aufweisen, damit gestohlene Chipkarten systemweit gesperrt werden können. Die dazu verwendeten Verfahren sind stark von der jeweiligen Anwendung und vom Systemdesign abhängig, können jedoch auf einige fundamentale Techniken zurückgeführt werden.

Um gefälschte oder verloren gegangene Chipkarten von der Benutzung auszuschließen, benötigt man Listen, in denen entweder die gültigen oder nicht mehr gültigen Karten anhand eines eindeutigen Merkmals aufgeführt sind. Üblicherweise ist dieses eine Nummer, also beispielsweise die Kartennummer. Vom Gesichtspunkt des tadellosen Systemdesigns her, bei dem alles, was nicht explizit erlaubt ist, implizit verboten ist, wären Listen mit allen gültigen Karten am besten. Diese so genannten Whitelists führen aber bei größeren Systemen zu unhandlich großen Aufstellungen und der Notwendigkeit sehr häufiger Aktualisierungen, was sich leicht daran verdeutlichen lässt, dass ein System mit 10 Millionen Chipkarten und einer 8 Byte langen Kartennummer zu einer Whitelist mit 80 MByte Daten führen würde.

Dies ist der Grund, warum in der Praxis so genannte Blacklists verwendet werden, in denen nur die gesperrten Karten aufgelistet sind. Im obigen Beispiel würde sich die Liste bei einem Anteil von einem Prozent gesperrter Karten auf 800 kByte reduzieren. Sollte es jedoch notwendig sein, bei einem System wesentlich mehr als ein Prozent aller Karten aufgrund von Angriffen oder Verlust zu sperren, werden auch mit diesem Modell die Listen schnell unhandlich groß.

Um den notwendigen Datentransfer und die Datenmenge zwischen der listenführenden Instanz und den listenprüfenden Instanzen noch weiter zu minimieren, existieren gelegentlich noch Redlists. In ihnen werden nachweislich gefälschte Chipkarten geführt, die entweder sofort einzuziehen oder zumindest für jede weitere Transaktion zu sperren sind. Die Anzahl der Einträge in dieser Liste liegt selbst bei großen Systemen nur im zwei- oder dreistelligen Bereich.

Chipkarten können bei Systemen, die online arbeiten, zeitaktuell gegen diese Listen geprüft werden. Bei vollständig oder teilweise offline arbeitenden Systemen muss die Blacklist und die Redlist so oft wie möglich zu den Terminals übertragen werden. Dieses sollte zumindest tagesaktuell geschehen, da sonst ein Schutzmechanismus basierend auf Sperrlisten nicht mehr greifen kann.

Angriff und Abwehr: Computerviren und trojanische Pferde

Computerviren waren bislang bei Chipkarten gänzlich unbekannt, da ein Nachladen von Programmcode in der Anwendungsphase technisch nicht vorgesehen war. Die modernen Chipkarten-Betriebssysteme besitzen jedoch Mechanismen, um Programmcode auch nach Ausgabe der Chipkarte an den Benutzer nachzuladen und auszuführen. Damit sind im

Prinzip die Voraussetzungen für Computerviren auf Chipkarten geschaffen. Der Definition nach sind Computerviren Programme, die sich selbst reproduzieren und damit weiterverbreiten können. Besitzen sie nicht die Eigenschaft der Fortpflanzung, nennt man sie nicht Viren, sondern trojanische Pferde. Beiden Arten ist aber gemein, dass sie unter Umständen auf ihrem Wirtsrechner unerlaubte Aktionen ausführen. Bei einer Chipkarte wäre dies beispielsweise das Lesen und Ausgeben geheimer Schlüssel.

Im Gegensatz zu normalen PCs ist es aber bei Chipkarten nicht ohne weiteres möglich, Programme in den Speicher zu laden und auszuführen. Es sind Sicherheitsmechanismen vorgesehen, die die unautorisierte Ausführung von Programmen verhindern. Dazu ist, abhängig von der jeweiligen Anwendung, beispielsweise vorab eine Authentisierung des Terminals notwendig. Zudem muss der Programmcode durchwegs zumindest mit einem MAC oder einer digitalen Signatur abgesichert sein. Zusätzlich findet bei manchen Chipkarten-Betriebssystemen eine gegenseitige Abschottung des Speichers der einzelnen Anwendungen durch Soft- oder Hardware statt, sodass sich Anwendungen in Chipkarten nicht gegenseitig beeinflussen können. Diese starken Sicherheitsmechanismen führen dazu, dass sowohl Computerviren als auch trojanische Pferde (die unbeabsichtigt während der Benutzung nachgeladen werden) wohl auf absehbare Zeit keine Anwendungen in ihrer Funktion und Sicherheit beeinträchtigen werden.

Angriff und Abwehr: Erschöpfende Schlüsselsuche

Ein möglicher Angriff auf kryptografischer Ebene ist die erschöpfende Schlüsselsuche. Der Angreifer benötigt dazu ein oder besser mehrere Klartext-Schlüsseltext-Paare und natürlich den verwendeten Kryptoalgorithmus. Dann verschlüsselt er den gegebenen Klartext mit allen möglichen Schlüsseln, bis er den ebenfalls bekannten Schlüsseltext erhält. Diesen gefundenen Schlüssel probiert er nun noch an allen anderen Klartext-Schlüsseltext-Paaren aus. Ist jedes Mal eine korrekte Verschlüsselung ausführbar, hat er mit einer sehr hohen Wahrscheinlichkeit den richtigen Schlüssel ermittelt. Dieses Verfahren ist prinzipiell für alle Verschlüsselungsalgorithmen geeignet, allerdings ist es nicht immer die schnellste Methode, die geheimen Schlüssel zu ermitteln.

Schon 1993 veröffentlichte Michael Wiener die Pläne für einen Spezialrechner, der eine Million Dollar kosten sollte und für ein gegebenes Klartext-Schlüsseltext-Paar innerhalb von sieben Stunden alle DES-Schlüssel durchprobieren konnte [Wiener 93]. Damit wäre es möglich, in durchschnittlich 3,5 Stunden einen 56 Bit langen DES-Schlüssel zu ermitteln. Einige Jahre später, 1997, wurde mit etwas mehr als 70000 über Internet verbundenen Computern in 97 Tagen durch systematische Suche der DES-Schlüssel für ein von der Firma RSA Inc. vorgegebenes Klartext-Schlüsseltext-Paar ermittelt [RSA 97]. Die Suchleistung betrug in der Endphase des Versuchs ca. 0,7 % des DES-Schlüsselraums pro 24 Stunden. Ein anderes Beispiel für die durch Zusammenschaltung von Rechnern im Internet erreichbare große Rechenleistung ist die SETI@Home Initiative zur Suche von extraterrestrischem Leben im All. Der als massiver Parallelrechner aufgebaute „DES Cracker“ der EFF hat 1998 zur Ermittlung eines unbekannten DES-Schlüssels nur noch 56 Stunden benötigt [EFF 98].

Um diesen Angriff abzuwehren, werden in der Praxis mehrere Wege beschritten. Der einfachste und allgemein bekannte ist der, wonach der Schlüsselraum des Kryptoalgorithmus so groß sein muss, dass eine systematische Suche innerhalb eines akzeptablen

Zeitraums und auch mit sehr großer Rechenleistung nicht mehr möglich ist. Dies ist der Grund, warum der DES mittlerweile prinzipiell durch den Triple-DES ersetzt wird. Der Schlüsselraum des DES ist im Verhältnis zur heute zur Verfügung stehenden Rechenleistung zu klein geworden.

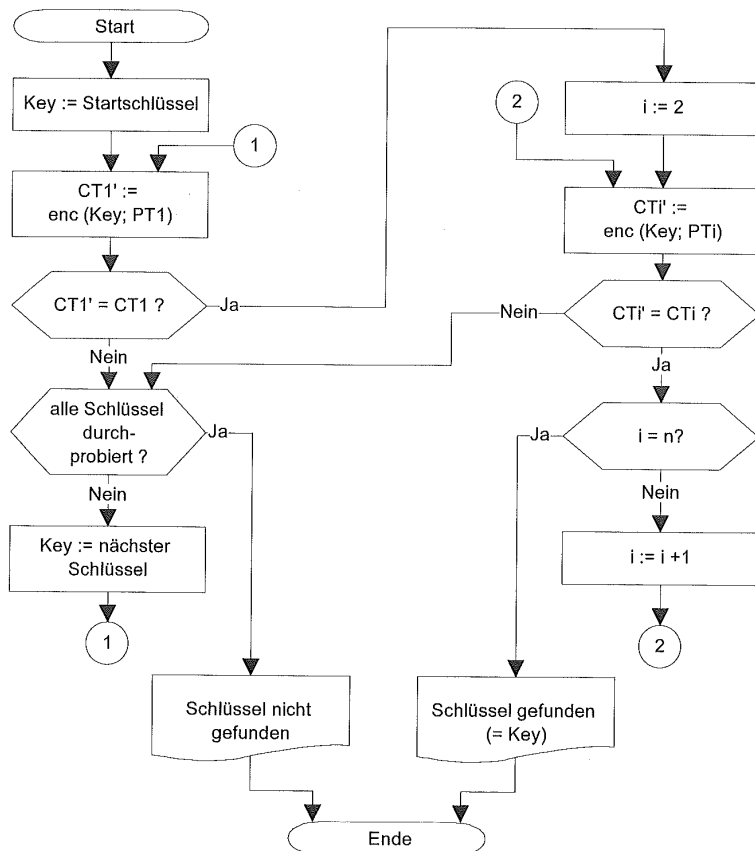


Bild 8.44 Grundsätzliche Vorgehensweise bei der erschöpfenden Schlüsselsuche mit einem gegebenen Kryptoalgorithmus und mehreren Klartext-Schlüsseltext-Paaren, nach einer Vorlage von James Massey [Massey 97]. Folgende Abkürzungen wurden benutzt: CT: Schlüsseltext (*ciphertext*); PT: Klartext (*plaintext*); CT_i, PT_i das Klartext-Schlüsseltext-Paar mit der Nummer *i*; *n*: Anzahl der Klartext-Schlüsseltext-Paare.

Eine weitere Abwehrmaßnahme kann sehr einfach geschaffen werden, indem man die Anwendungsprotokolle so aufbaut, dass keine Paare von Klartext und Schlüsseltext mehr vorkommen. In den meisten Fällen müssen nämlich bei Chipkarten-Anwendungen überhaupt keine Daten verschlüsselt, sondern lediglich mit einem MAC abgesichert werden. Da die Abbildung von mehreren Klartextblöcken auf einem MAC nicht eindeutig ist, wird

der Brute-force-Angriff für einen Angreifer ungleich aufwendiger als mit einem gegebenen Klartext-Schlüsseltext-Paar.

Stellt man in der Chipkarte dem Klartext eine Zufallszahl voran (*salt*) und verschlüsselt erst dann die übergebenen Daten bzw. bildet den MAC darüber, ändern sich bei jedem Aufruf dieser Funktion die zu verschlüsselnden Daten, und das Ergebnis ist ebenfalls unterschiedlich. Dies erschwert auch die erschöpfende Schlüsselsuche, da die Zufallszahl in vielen Fällen nicht öffentlich sein muss. Sie könnte beispielsweise ein Geheimnis zwischen Chipkarte und Sicherheitsmodul sein. Im Übrigen schützt eine den zu verschlüsselnden Daten innerhalb der Chipkarte zusätzlich vorangestellte Zufallszahl (auch wenn sie öffentlich ist) sehr gut gegen den Angriff durch differenzielle Fehleranalyse (DFA) und gegen Leistungsanalyse (SPA/DPA).

Man kann einem potenziellen Angreifer das Leben noch etwas schwerer machen, wenn man dynamische Schlüssel (*session keys*) verwendet, die sich bei jeder Verschlüsselungsoperation ändern. Selbst wenn es nun der Angreifer durch einen glücklichen Zufall schafft, den Schlüssel zu ermitteln, kann er keinen Nutzen daraus ziehen, da dieser Schlüssel bei der nächsten Transaktion schon wieder geändert ist.

9 Qualitätssicherung und Test

9.1	Test der Kartenkörper	578
9.2	Test der Hardware von Mikrocontrollern	584
9.3	Evaluierung und Test von Software	585
9.3.1	Evaluierung	586
9.3.2	Testmethoden für Software	593
9.3.3	Dynamische Tests von Betriebssystemen und Anwendungen	601

Gerade bei Chipkarten sind Qualitätssicherung und die dazugehörigen Testprozeduren und -verfahren von besonderer Bedeutung. Ein Chipkarten-Hersteller muss ein Produkt in sehr hohen Stückzahlen, großer Qualität und niedrigem Preis fertigen. Im Gegensatz zu anderen Industriezweigen enthält das Produkt noch dazu einen relativ aufwendigen und empfindlichen Mikrocontroller sowie Software, die größtenteils nicht mehr geändert werden kann.

Vergleicht man diese Ausgangssituation beispielsweise mit Standardsoftware für PCs, so sieht man den grundlegenden Unterschied sehr schnell. Dort hat es sich seit einigen Jahren regelrecht eingebürgert, dass die erste Auslieferung (meistens mit einer „0“ am Ende der Versionsbezeichnung) innerhalb einiger Wochen bis maximal ein bis zwei Monaten von revidierten und verbesserten Versionen (mit den Endungen „a“, „b“, „c“ usw.) abgelöst wird. Im Chipkartenbereich wäre dies unmöglich. Die maskenprogrammierte Software kann vom Speicherprinzip her nicht mehr geändert werden, und eine größere Anzahl ausgegebener Karten kann ebenfalls nicht in einer wie immer gearteten Rückrufaktion umgetauscht werden. Selbst bei Karten, die nicht im besonders sensiblen Zahlungsverkehr eingesetzt werden, wäre der Ruf des Kartenherausgebers über Jahre hinweg stark geschädigt und die entstehenden Kosten immens hoch.

Deshalb ist die Qualitätssicherung und das Testing bei der Produktion von Chipkarten von essenzieller Wichtigkeit. Nach Herstellung und Auslieferung der Karten ist es hier eben unmöglich, kurze Zeit später eine verbesserte Version „nachzuschieben“. Dies führt natürlich zu einem hohen Aufwand, um das Produkt so fehlerarm wie möglich herzustellen. Bei den unterschiedlichen Tests muss grundsätzlich zwischen Tests für die Qualifikation und die Produktion unterschieden werden. Testen für die Qualifikation bedeutet, dass hier eine grundlegende Aussage getroffen wird, ob die betreffende Chipkarte überhaupt verwendbar ist. Diese Tests finden üblicherweise vor der Markteinführung eines neuen Kartenkörpers, Chips, Moduls oder Betriebssystems statt. Erfüllt das neue oder das veränderte Produkt die gestellten Anforderungen, ist es damit für die Produktion „qualifiziert“ und kann in großen Stückzahlen hergestellt werden. Dort werden dann die Qualifikationstests nur mehr stichprobenartig in großen zeitlichen Abständen durchgeführt.

Für die Tests in der Produktion benutzt man andere Prüfmethoden. Diese sind durchweg alle in kurzer Zeit und unkompliziert durchführbar, so dass die unabdingbaren Forderungen einer Massenfertigung nach kleiner Durchlaufzeit und hohem Durchsatz erfüllt werden können. Meist werden für diese Tests nur einfache mechanische und elektrische Rahmeparameter gemessen sowie geeignete Prüfkommandos an den Mikrocontroller der Chipkarte gesandt.

Vor allem in Hinblick auf Interoperabilität zwischen Chipkarten und Terminals ausgelegt sind viele Testspezifikationen für große Chipkarten-Anwendungen. Ein gutes Beispiel

dafür ist die GSM 11.17 namens „Subscriber Identity Module (SIM) Test Specification“ mit ca. 100 Seiten Umfang. Darin werden vom Kartenkörper, über elektrische Rahmenparameter (Spannung, Stromaufnahme u. Ä.) bis hin zu Übertragungsprotokoll, Kommandos und Dateien detailliert Tests für GSM-Chipkarten beschrieben. Die Tests der GSM 11.17 sind dabei folgendermaßen gegliedert:

- physikalische Eigenschaften
- elektrische Signale und Übertragungsprotokolle
- logisches Modell
- Sicherheitsfunktionen
- Funktionen
- Kommandos
- Inhalt von Dateien

Die Aufteilung der einzelnen Tests ist in dieser Norm ebenfalls sehr zweckmäßig und übersichtlich gestaltet. Ein einzelner Test besteht aus vier Teilen. Im ersten Teil ist der Test formal definiert und seine Anwendung spezifiziert. Danach folgen die zu erfüllenden Anforderungen und im dritten Teil der detaillierte Zweck des Tests. Zum Schluss wird dann die eigentliche Vorgehensweise beim Test festgelegt.

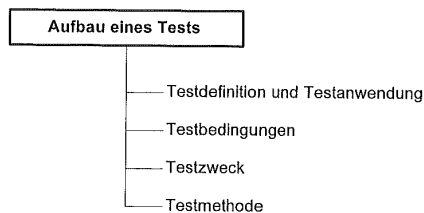


Bild 9.1 Genereller Aufbau eines Tests nach GSM 11.17. Diese Struktur ist so allgemein gehalten, dass sie im Prinzip für alle Chipkartentests gelten kann.

9.1 Test der Kartenkörper

Für den Test von Karten mit und ohne Chip gibt es momentan nur eine internationale Norm, die ISO/IEC 10 373. Auf europäischer Ebene existiert noch die EN 1292, die sich jedoch ausschließlich mit Chipkarten und Terminals und deren elektrischen Rahmenbedingungen beschäftigt. Zusätzlich findet man jedoch oft auch in Normen für Karten einzelne Tests und Testverfahren, um dort festgelegte Eigenschaften zu prüfen.

Auf den folgenden Seiten sind viele der üblichen Tests und Prüfungen für Karten im Überblick und alphabetischer Reihenfolge aufgeführt. Die Prüflabors der Kartenhersteller haben in der Regel ein Repertoire von 120 bis 150 unterschiedlichen Tests für Karten.

Grundlegend sind normierte Umweltbedingungen für die Testumgebung, was bedeutet, dass eine Temperatur von $23\text{ °C} \pm 3\text{ °C}$ und eine relative Luftfeuchtigkeit zwischen 40 % und 60 % im Testlabor herrschen muss. Die zu testenden Karten müssen mindestens 24 Stunden vor dem eigentlichen Test diesen Bedingungen entsprechend klimatisiert werden.

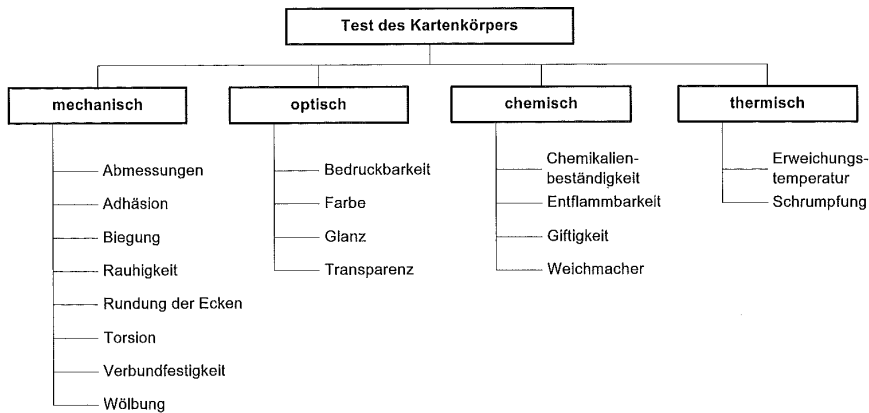


Bild 9.2 Klassifizierungsbaum mit einer Auswahl üblicher Tests des Kartenkörpers. Für die einzelnen Kartenelemente (Hologramm, Magnetstreifen, Chip u. a.) sind jeweils eine Reihe darauf zugeschnittener Tests notwendig.

Abnutzung des Magnetstreifens (*wear test for magnetic stripe*)

(Grundlage: ISO 7811-2; Prüfvorschrift: ISO/IEC 10373)

Um das magnetische Verhalten des Magnetstreifens bei Abnutzung zu ermitteln, werden zunächst die Testdaten geschrieben. Dann führt man einen mit 1,5 N belasteten Dummy-Schreib-/Lesekopf in der Härte 110 HV bis 130 HV und einem Krümmungsradius von 10 mm 1000mal über den Magnetstreifen (jeweils hin und zurück). Anschließend liest man die Testdaten. Die Signalamplitude muss sich in den Grenzen von ISO 7811-2 bewegen.

Adhäsion (*adhesion or blocking*)

(Grundlage: ISO 7810; Prüfvorschrift: ISO/IEC 10373)

Mit diesem Test wird überprüft, ob sich das Verhalten von Karten bei Lagerung unter bestimmten Umweltbedingungen ändert. Dazu werden fünf nicht hochgeprägte Karten aufeinander gelegt und 48 Stunden gleichförmig mit 2,5 kPa bei 40 °C und 90 % relativer Luftfeuchtigkeit beschwert. Anschließend untersucht man diese Karten auf Delaminierung, Entfärbung, Änderungen der Oberfläche und weitere visuell erkennbare Veränderungen.

Biegesteifigkeit (*bending stiffness*)

(Grundlage: ISO 7810; Prüfvorschrift: ISO/IEC 10373)

Um die erforderliche Biegesteifigkeit zu ermitteln, wird die Karte an der schmalen linken Seite mit der Vorderseite nach unten auf 3 Millimeter Tiefe fest eingespannt und zuerst die Biegung ohne Belastung gemessen. Anschließend belastet man die äußerste Kante mit einer Kraft von 0,7 N und berechnet die Differenz zur unbelasteten Durchbiegung. Das Ergebnis ist eine Aussage über die Biegesteifigkeit.

Der Test der Biegesteifigkeit wird oft auch bei Temperaturen unterhalb oder oberhalb der üblichen Testtemperatur von 23 °C durchgeführt.

Chemikalienbeständigkeit (*resistance to chemicals*)

(Grundlage: ISO 7810, ISO 7811-2; Prüfvorschrift: ISO/IEC 10 373)

Die Chemikalienbeständigkeit von Kartenkörper und Magnetstreifen wird mit den folgenden Tests ermittelt. Verschiedene Karten werden in die genau festgelegten Flüssigkeiten mit einer Temperatur zwischen 20 °C und 25 °C gelegt:

- in 5-prozentiges Salzwasser
- in 5-prozentige Essigsäure
- in 5-prozentige Natriumcarbonatlösung
- in wässrige Lösung mit 60 Prozent Ethanol
- in 10-prozentige Lösung von Zuckerwasser
- in Benzin (nach ISO 1817)
- in wässrige Lösung mit 50 Prozent Ethylenglykol

Nach einer Minute wird die Karte aus der entsprechenden Lösung genommen und visuell begutachtet bzw. der Dateninhalt des Magnetstreifens geprüft.

Dynamischer Biegetest (*dynamic bending stress*)

(Grundlage: ISO 7816-1; Prüfvorschrift: ISO/IEC 10373)

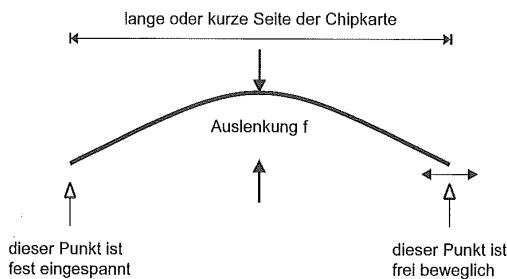


Bild 9.3 Schema der Beanspruchung einer Chipkarte beim Biegetest.

Der dynamische Biegetest ist in Bild 9.3 dargestellt. Mit einer Frequenz von 30 Biegungen pro Minute (= 0,5 Hz) wird die Karte 2 cm in der Länge bzw. 1 cm in der Breite gewölbt (f). Eine Karte muss bei diesem Test mindestens 250 Biegungen in jeder der vier möglichen Richtungen (insgesamt also 1000 Biegungen) ohne Schäden überstehen.

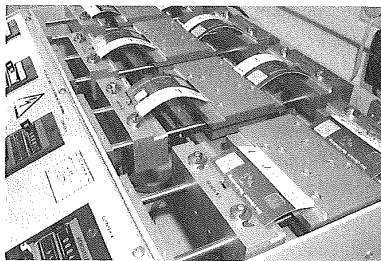


Bild 9.4 Maschine für den Test auf Biegung von Chipkarten.

Dynamischer Torsionstest (*dynamic torsional stress*)

(Grundlage: ISO 7816-1; Prüfvorschrift: ISO/IEC 10373)

Beim dynamischen Torsionstest wird die Karte $\pm 15^\circ$ um die Längsachse mit einer Frequenz von 30 Biegungen pro Minute ($= 0,5 \text{ Hz}$) belastet. Der Standard verlangt 1000 Torsionen, ohne dass die Chipfunktionalität ausfällt oder mechanische Schäden an der Karte sichtbar werden.

Elektrischer Widerstand der Kontakte

(*electrical resistance and impedance of contacts*)

(Grundlage: ISO 7816-1/2; Prüfvorschrift: ISO/IEC 10373)

Der elektrische Widerstand der Kontakte ist ein wichtiges Kriterium für die sichere elektrische Versorgung und die Datenübertragung zum Mikrocontroller der Chipkarte. Der Widerstand wird mit zwei Testkontakten gemessen, die an den beiden entgegengesetzten Ecken einer minimal dimensionierten Kontaktfläche mit $0,5 \text{ N} \pm 0,1 \text{ N}$ aufgedrückt werden. Der Widerstand zwischen den beiden vergoldeten und vorne mit $0,4 \text{ mm}$ Radius abgerundeten Testkontakten muss kleiner als $0,5 \Omega$ sein.

Elektromagnetische Felder (*electromagnetic fields*)

(Grundlage: ISO 7816-1; Prüfvorschrift: ISO/IEC 10373)

Bei diesem Test wird die Karte in ein statisches elektromagnetisches Feld der Stärke 1000 Oe ($= 79580 \text{ H}$) mit maximal 1 cm/s Geschwindigkeit eingeführt. Dabei darf sich der Speicherinhalt der Karte nicht ändern.

Entflammbarkeit (*flammability*)

(Grundlage: ISO 7813; Prüfvorschrift: ISO/IEC 10373)

Die Entflammbarkeit einer Karte wird gemessen, indem man sie an einer Kante im Winkel von 45° für 30 Sekunden in die definierte Flamme ($\varnothing 8,5 \text{ mm}$, Höhe: 25 mm) eines Bunsenbrenners hält.

Feldwechsel im Magnetstreifen (*flux transition spacing variation*)

(Grundlage: ISO 7811-2; Prüfvorschrift: ISO/IEC 10373)

Mit diesem Test stellt man fest, ob der Magnetisierungswechsel in der Codierung der einzelnen Bits im Magnetstreifen gleichmäßig und ausreichend ist. Dazu führt man einen Lesekopf über den Magnetstreifen und zeichnet den Feldwechsel auf. Anschließend vergleicht man den gemessenen Istwert mit dem Sollwert in ISO 7811-2.

Höhe der Prägung (*embossing relief height of character*)

(Grundlage: ISO 7811-1; Prüfvorschrift: ISO/IEC 10373)

In diesem Test misst man die Dicke der Karte an den Stellen der Hochprägung mit einem Mikrometer mit einer Anpresskraft von $3,5 \text{ N}$ bis $5,9 \text{ N}$.

Höhe und Oberflächenprofil des Magnetstreifens

(*height and surface profil of the magnetic stripe*)

(Grundlage: ISO 7811-2/4/5; Prüfvorschrift: ISO/IEC 10373)

Diese Prüfung dient dazu, sowohl die Höhe als auch die Ebenheit des Magnetstreifens zu vermessen. Dazu verwendet man ein in der Norm detailliert beschriebenes, spezielles Messgerät, mit dem ein Höhenprofil erstellt werden kann.

Kartenformat und Kartendicke (*dimensions of cards*)

(Grundlage: ISO 7810; Prüfvorschrift: ISO/IEC 10373)

Mit diesem Test wird Höhe, Breite und Dicke einer nicht hoch geprägten Karte gemessen. Die Karte wird mit 2,2 N belastet und dann in Höhe und Breite mit einem Profilprojektor vermessen. Zur Dickenmessung wird die Karte in vier gleiche Rechtecke eingeteilt und jeweils in der Mitte mit einem Mikrometer mit einer Anpresskraft zwischen 3,5 N und 5,9 N gemessen. Sowohl der gemessene Maximal- als auch der Minimalwert wird mit der genormten Dicke verglichen.

Kartenformat und Kartenwölbung unter Einfluss von Temperatur und Luftfeuchtigkeit (*card dimensional stability and warpage with temperature and humidity*)

(Grundlage: ISO 7810; Prüfvorschrift: ISO/IEC 10373)

Da sich bestimmte Kunststoffe in ihrer Form und Größe durch den Einfluss von Luftfeuchtigkeit stark verändern, muss man die Einhaltung der Normen auch unter diesen Bedingungen durch entsprechende Tests überprüfen.

Dazu legt man eine Karte plan auf eine Unterlage und variiert Temperatur und Luftfeuchtigkeit. Die folgenden vier Rahmenbedingungen kommen zur Anwendung:

– 35 °C, + 50 °C, + 25 °C bei 5 % relativer Luftfeuchtigkeit und + 25 °C bei 95 % relativer Luftfeuchtigkeit. Nach jeweils 60 Minuten unter diesen Bedingungen werden Größe und Kartenwölbung mit den bekannten Messverfahren überprüft.

Lage der Kontakte (*location of contacts*)

(Grundlage: ISO 7816-2; Prüfvorschrift: ISO/IEC 10373)

Um die Lage der Kontakte zu vermessen, benutzt man diesen Test, bei dem die Chipkarte auf eine plane Fläche gelegt und mit $2,2 \text{ N} \pm 0,2 \text{ N}$ belastet wird. Anschließend werden die Kontakte mit einer beliebigen Messmethode, die eine maximale Ungenauigkeit von 0,05 mm hat, in Bezug auf die Kartenkanten gemessen.

Lichtdurchlässigkeit (*light transmittance*)

(Grundlage: ISO 7810; Prüfvorschrift: ISO/IEC 10373)

Es gibt Karten, die auf einer Inlettfolie einen optischen Barcode besitzen. Um nun die Lichtdurchlässigkeit der Deckfolien und des restlichen Kartenkörpers zu bestimmen, gibt es ebenfalls einen entsprechenden Test. Dazu wird die Karte mit einer Lichtquelle angeleuchtet und mit einem Detektor auf der gegenüberliegenden Seite die Transmission des Lichtes mit einem bis 900 nm empfindlichen Detektor gemessen.

Oberflächenprofil der Kontakte (*surface profile of contacts*)

(Grundlage: ISO 7816-1/2; Prüfvorschrift: ISO/IEC 10373)

Mit dieser Prüfvorschrift wird das Oberflächenprofil der einzelnen Kontakte zur Oberfläche der gesamten Karte in Beziehung gesetzt. Damit soll sichergestellt werden, dass die Kontakte annähernd in einer Ebene mit dem Kartenkörper liegen.

Oberflächenrauheit des Magnetstreifens (*surface roughness of the magnetic stripe*)

(Grundlage: ISO 7811-2; Prüfvorschrift: ISO/IEC 10373)

Um die Oberflächenrauigkeit des Magnetstreifens zu messen, benutzt man das Gleiche Messgerät wie für Höhe und Oberflächenprofil. Allerdings wird es mit einem speziellen Tastkopf ergänzt, sodass die Rauigkeit des Magnetstreifens erfasst werden kann. Dieser Test ist wichtig, weil die Oberflächenrauigkeit einer der wesentlichen Faktoren für die Abnutzung von Schreib-/Leseköpfen bei Magnetstreifenlesern ist.

Röntgenstrahlen (*X-rays*)

(Grundlage: ISO 7816-1; Prüfvorschrift: ISO/IEC 10373)

Ähnlich wie bei UV-Licht verändern sich die Speicherinhalte von (E)EPROM-Zellen bei Bestrahlung mit Röntgenstrahlen. Um die Resistenz des Speichers zu prüfen, bestrahlt man den Chip mit Röntgenstrahlung der Energie von 70 kV. Anschließend prüft man auf eine Veränderung der Speicherinhalte, bzw. ob sich der Speicher wieder beschreiben lässt.

Schwingungen (*vibrations*)

(Grundlage und Prüfvorschrift: ISO/IEC 10373)

Da Karten während des Transports und der Benutzung oft starken Schwingungen ausgesetzt sind (z. B. Mobiltelefon im Auto), wurde hierfür ebenfalls eine Prüfvorschrift erstellt. Die Karte muss danach auf einem Schwingungstisch jeweils in allen drei Achsen mit einer Amplitude von maximal 1,5 mm und einem Frequenzdurchlauf von 10 Hz bis 500 Hz getestet werden. Dabei darf es weder Einbußen hinsichtlich der Chipfunktionalität noch in den Speicherinhalten geben.

Signalamplitude des Magnetstreifens (*amplitude measurement*)

(Grundlage: ISO 7811-2; Prüfvorschrift: ISO/IEC 10373)

Mit dieser Messung wird die Signalamplitude und die Auflösung der Magnetstreifencodierung überprüft. Dazu benutzt man einen genormten Schreib-/Lesekopf, der mit einer genau festgelegten Geschwindigkeit über die Magnetpiste geführt wird.

Statische Elektrizität (*static electricity*)

(Grundlage: ISO 7816-1; Prüfvorschrift: ISO/IEC 10373)

Mit diesem Test, der nur bei Chipkarten sinnvoll ist, wird die Festigkeit des Chips gegenüber elektrostatischen Entladungen, d. h. ESD-Festigkeit (*electrostatic discharge – ESD*), geprüft. Dazu wird ein 100 pF Kondensator mit jeweils ± 1500 Volt aufgeladen und über einen strombegrenzenden Widerstand mit $1500\ \Omega$ über die verschiedenen Kontaktflächen des Chips entladen. Dieser darf dabei weder in seinen Funktionseinheiten Schaden nehmen noch in seinen Speicherinhalten verändert werden.

UV-Licht (*ultraviolet light*)

(Grundlage: ISO 7816-1; Prüfvorschrift: ISO/IEC 10373)

Da (E)EPROM-Speicher ihren Inhalt bei Bestrahlung mit UV-Licht verlieren, gibt es einen speziellen Test, um die Unempfindlichkeit dagegen sicherzustellen. Die Chipkarte wird zwischen 10 und 30 Minuten mit UV-Licht der Wellenlänge 254 nm und einer Energie von 15 W/cm^2 bestrahlt. Dabei darf sich der Dateninhalt im (E)EPROM nicht verändern.

Verbundfestigkeit von Mehrschichtkarten (*delamination*)

(Grundlage: ISO 7810; Prüfvorschrift: ISO/IEC 10373)

Der folgende Test ist nur zweckmäßig bei Mehrlagenkarten, die bekanntermaßen aus laminierten Kunststofffolien zusammengesetzt sind. Mit einem scharfen Messer trennt man die Deckfolie von den Kernfolien an einer Stelle ab. Ausgehend von dem abgetrennten Stück versucht man nun, die beiden laminierten Folien auseinander zu ziehen. Dabei misst man die dazu erforderliche Kraft und vergleicht sie mit Referenzwerten.

Wölbung (*card warpage*)

(Grundlage: ISO 7810; Prüfvorschrift: ISO/IEC 10373)

In diesem Test misst man die Wölbung der Karte. Dazu wird sie auf eine ebene Platte gelegt und mit einem Profilprojektor die Kartenwölbung vermessen. Dieser Test ist vor allem für Karten gedacht, die aus einer aufgerollt angelieferten Folie ausgestanzt werden.

Natürlich gibt es noch viele weitere Tests, wie beispielsweise die Anzahl der Steckzyklen, Farbabriebfestigkeit, Weichmacherstabilität, Schweiß- und Speichelechtheit. Je nach Einsatzgebiet der Karte müssen die entsprechenden Tests dann ausgewählt und durchgeführt werden.

9.2 Test der Hardware von Mikrocontrollern

Bei der Qualitätssicherung muss neben dem Kartenkörper vor allem die Funktionsfähigkeit des Mikrocontrollers sichergestellt werden, welche bei modernen Chipkarten die wichtigste und auch anfälligste Komponente ist.

Schon bei der Herstellung der Halbleiter werden mit entsprechenden Tests die CPU und die Speicher auf dem Mikrocontroller in allen Varianten geprüft. Um diese Tests durchführen zu können, besitzt jeder Mikrocontroller ein so genanntes Test-ROM, das diverse Programme für den Testzugriff auf CPU und Speicher von außen besitzt. Zusätzlich gibt es manchmal auch noch besondere Anschlüsse, die wahlfreie Zugriffe auf die zentralen Busse des Mikrocontrollers zulassen. Mit Nadeladaptoren werden dann während der Halbleiterfertigung die entsprechenden Pads auf dem Chip kontaktiert und die notwendigen Prüfprogramme ausgeführt. Damit diese Prüfanschlüsse zu einem späteren Zeitpunkt nicht für Angriffe genutzt werden können, schneidet man sie während der Vereinzelung der Chips in der Fertigung ab. Damit ist es dann bei dieser Art von Chips von außen nicht mehr möglich, einen Zugriff auf den internen Bus zu erhalten.

Nachdem die Dies in Module verpackt worden sind, findet über die Modulkontakte natürlich ebenfalls eine Prüfung statt. Dazu führt man oft nur eine Anhaltsequenz aus und prüft, ob ein ATR empfangen werden kann. Ist dies möglich, geht man davon aus, dass der Halbleiter während der Modulverpackung keinen größeren Schaden genommen hat und alle Bonddrähte korrekt angeschlossen wurden. Einen ähnlichen „ATR-Test“ benutzt man auch direkt nach der Implantierung der Module in die Kartenkörper. Der Zweck ist festzustellen, ob die kurzzeitige Temperaturerhöhung beim Einbau in den Kartenkörper das Modul geschädigt hat.

Vor der Initialisierung der Chipkarten wird dann der Mikrocontroller sehr genau getestet. Dazu benutzt man die in diesem Verarbeitungsschritt freigegebenen Testkomman-

dos.¹ Nach einem erfolgreichen Verlauf der 10 bis 100 Sekunden dauernden Prüfprogramme werden die Kommandos für die weitere Benutzung irreversibel gesperrt. Diese zeitaufwendigen Tests können in diesem Produktionsschritt deshalb ohne Beeinträchtigung des Durchsatzes ausgeführt werden, da die zum Einsatz kommenden Initialisierungsmaschinen massiv parallel arbeiten und deshalb die Testzeit nicht stark zum Tragen kommt. Beispielsweise prüft man hier, ob sich alle Bytes im EEPROM schreiben und wieder löschen lassen und ob das RAM vollständig in Ordnung ist. Ein während des Bondings entstandener Kratzer auf dem Chip könnte dazu führen, dass sich einige Zellen im EEPROM nicht mehr korrekt beschreiben lassen oder dass bestimmte Bereiche im ROM einen fehlerhaften Inhalt haben.

Nach der Initialisierung und Personalisierung der Karten führt man je nach Hersteller noch verschiedene Endtests durch. Üblicherweise benutzt man dazu vollautomatische Testwerkzeuge, die sich selbst kalibrieren. Sie lesen also alle für den Test relevanten Daten aus der Chipkarte aus und prüfen dann auf dieser Datengrundlage.

Neben den relativ einfachen und schnell durchführbaren Tests, die alle produzierten Karten durchlaufen, gibt es aber noch Stichprobentests, die nur an einzelnen Karten durchgeführt werden. Bei diesen der laufenden Produktion entnommenen Karten können natürlich gegebenenfalls auch zerstörende Prüfungen vorgenommen werden.

Für die Qualifizierung und für laufende Stichproben wurde auch die Norm EN 1292 veröffentlicht, die viele unterschiedliche Testverfahren für den Mikrocontroller definiert. Typische Stichproben- und Qualifikationstests für Mikrocontroller sind:

- Anstiegs- und Abfallzeiten am I/O-Kontakt (EN 1292)
- Anzahl der möglichen Schreib-/Löschzyklen des EEPROMs
- Datenerhalt im EEPROM
- Detektion von Über- und Unterfrequenz an CLK
- Detektion von Über- und Unterspannung an Vcc
- Spannung am I/O-Kontakt (EN 1292)
- Stromaufnahme an CLK (EN 1292)
- Stromaufnahme an Reset-Kontakt (EN 1292)
- Stromaufnahme an Vcc (EN 1292)
- Stromaufnahme an Vpp (EN 1292)

Zusätzlich verwendet jeder Kartenhersteller natürlich auch noch eigene Prüfungen, um damit spezielle Eigenschaften des jeweils eingebauten Mikrocontrollers abzudecken. So gibt es beispielsweise eigene Tests für die verschiedenen Sensoren auf dem Chip, damit auch diese entsprechend geprüft werden können.

9.3 Evaluierung und Test von Software

Das Prüfen von physikalischen vorhandenen Bauteilen, wie es Kartenkörper und Module von Chipkarten darstellen, kann weitgehend mit den traditionell üblichen Verfahren durchgeführt werden. Die elektrischen Kennwerte lassen sich mit automatischen Testmaschinen ebenfalls sehr gut ermitteln. Bei der Software in den Mikrocontrollern sieht es aber etwas

¹ Siehe auch Abschnitt 7.11 „Befehle zum Test der Hardware“.

anders aus. Die Prüfung von Software auf Fehler ist zwar seit den ersten Programmen in den letzten 40 Jahren stetig verfeinert worden, und es gibt viele anerkannt gute Verfahren zur Erzeugung fehlerarmer Programme, doch in der täglichen Praxis sind Softwarefehler dennoch relativ häufig vorzufinden.

Bei den meisten Anwendungen ist dies nicht problematisch, da durch die Ausgabe einer revidierten Version in kurzer Zeit der Fehler neutralisiert werden kann. Bei Chipkarten ist dies aber nicht so leicht möglich, da sich die Software größtenteils im ROM des Mikrocontrollers befindet. Eine neue Version benötigt einen kompletten Fertigungsdurchlauf beim Halbleiterhersteller, der in einer zeitlichen Größenordnung von 8 bis 12 Wochen liegt. Sind die Chipkarten erst einmal den Benutzern übergeben, ist es praktisch unmöglich, Änderungen an der bestehenden Software durchzuführen. Aus diesen sehr strengen Rahmenbedingungen folgt, dass Software für Chipkarten-Mikrocontroller äußerst „fehlerarm“ sein muss. Der Begriff „fehlerfrei“ wäre noch besser, doch mit dem derzeitigen Stand der Softwareerstellung ist dies nicht annähernd zu erreichen.

Die Thematik des Testens von Software ist bekanntermaßen äußerst umfangreich und in vielen Büchern in allen ihren Varianten und Richtungen beschrieben. Hier kann nur ein kurzer Abriss über dieses in der Informatik mittlerweile fast schon eigenständige Fachgebiet gegeben werden. Deshalb wird im Folgenden auch nur auf die Prüfung von Software für Chipkarten-Mikrocontroller eingegangen, die einige Besonderheiten auszeichnet. Stellvertretend für die umfangreiche Fachliteratur sei hier nur Glenford J. Myers genannt [Myers 95], der eine gute Einführung in das Thema gibt. Zusätzlich sei hier noch darauf hingewiesen, dass gerade im Bereich der militärischen Normung viele gute und langfristig erprobte Verfahren zur Erzeugung und Prüfung von Software als Norm erhältlich sind.

9.3.1 Evaluierung

Aufgrund der Eigenschaft, Daten sicher aufbewahren zu können, werden Chipkarten vor allem in sicherheitssensiblen Bereichen eingesetzt. Doch liegen die Vorteile von Chipkarten nicht nur in der sicheren Aufbewahrung von Daten, sondern ebenso im Bereich der gesicherten Ausführung von kryptografischen Algorithmen.

Gerade der Bereich des elektronischen Zahlungsverkehrs ist ein expandierender Markt für Chipkarten. Weil hier bei großen Systemen enorme Geldströme fließen, muss der jeweilige Anwendungsanbieter oder Kartenherausgeber sehr großes Vertrauen in den Halbleiterhersteller, Betriebssystem-Hersteller und auch den Personalisierer der Chipkarten haben. Der Anwendungsanbieter muss absolut sicher sein können, dass die Software in der Chipkarte fehlerfrei die geforderten Transaktionen für den Zahlenverkehr ausführt und dass keine Sicherheitslücken oder gar absichtlich in die Software eingebrachten Falltüren vorhanden sind.

Beispielsweise könnte man mit einem geheimen Kommando an die Chipkarte die PIN und alle geheimen Schlüssel auslesen. Im Falle einer GSM- oder ec-Karte wäre dann der vermeintliche Angreifer fähig, beliebig viele Karten zu klonen und dann voll funktionsfähig zu verkaufen.

Doch die Sicherheitsforderungen betreffen nicht nur die Herstellung der Chipkarte, sondern ebenso Initialisierung und Personalisierung, denn dort werden die geheimen Schlüssel

und die PIN in die Chipkarte geladen. Der Kartenherausgeber muss dem Lieferanten der Karte ein sehr hohes Vertrauen in Fragen der Sicherheit entgegenbringen.

Das Gleiche gilt auch für die grundsätzliche Sicherheit der Software in der Chipkarte. Es muss nicht absichtlich eine Falltür zum Ausspähen von Informationen eingebaut worden sein. Jedoch könnte es durch eine Fehlfunktion sehr wohl möglich sein, mit einer im normalen Ablauf nicht verwendeten Kommandokombination Informationen aus einer Karte auszulesen oder zu schreiben. Die Wahrscheinlichkeit für einen solchen Zufall ist zwar äußerst gering, doch bekanntermaßen ist es beim derzeitigen Stand der Softwaretechnologie nicht möglich, die Fehlerfreiheit eines Programms in allen Fällen zu garantieren. In Zukunft werden mit Sicherheit Firmen, die Software für Chipkarten erstellen, mit solchen juristischen Formulierungen nicht mehr einfach alle Verantwortung ablegen können.

Um ein Produkt auf seine Vertrauenswürdigkeit zu prüfen, bleiben dem Anwendungsanbieter nur zwei Möglichkeiten: Entweder prüft er die Chipkarte mit der Software in allen Variationen selbst, oder er lässt sie von einer vertrauenswürdigen Instanz prüfen. Der erste Weg, die Prüfung der Chipkarte beim Anwendungsanbieter, ist oft nur sehr eingeschränkt möglich, da dieser zum Teil nicht das technische Fachwissen und die technischen Möglichkeiten für die Tests besitzt. Die zweite Möglichkeit, die Vergabe der Tests an eine weitere Instanz, gilt momentan als eine für alle Beteiligten tragbare Lösung.

Die gleiche Problematik gibt es schon seit vielen Jahren in der Software- und Systementwicklung im Militärbereich. Sie ist also in der Chipkartenwelt nicht neu. Um die Vertrauenswürdigkeit von Softwareprodukten objektiv „messbar“ zu machen, d. h. eine Metrik dazu festzulegen, wurde im Jahre 1983 vom NCSC (US National Computer Security Center) ein Kriterienkatalog zur Bewertung der Vertrauenswürdigkeit von informationstechnischen Systemen herausgegeben. Das NCSC wurde 1981 vom nordamerikanischen Department of Defense (DoD) gegründet. Im Jahre 1985 wurden dann die TCSEC (*Trusted Computer System Evaluation Criteria*) veröffentlicht. Das Buch hatte einen orangefarbenen Einband, weshalb es auch allgemein als Orange Book bekannt ist. Diese Kriterien dienen dem NCSC als Richtlinien zur Evaluierung von informationstechnischen Systemen.

Die TCSEC hat weltweit für praktisch alle Kriterienkataloge im Sektor der Informationstechnik Vorbildfunktion. In Europa wurden – allerdings auch auf der Grundlage von TCSEC – eigene Kriterien definiert. Sie wurden als ITSEC (*Information Technique System Evaluation Criteria*) im Jahre 1990 erstmals veröffentlicht und liegen seit 1991 in einer revidierten Version vor.

Um einen einheitlichen Standard zur Prüfung von Software auf Korrektheit zur Verfügung zu haben, wurden die Common Criteria (CC) geschaffen, die sozusagen die Essenz aus TCSEC und ITSEC sind. Die Common Criteria sind auch besser für die Evaluierung von Software ausgerichtet, als dies bei TCSEC und ITSEC der Fall ist. Obwohl bereits 1996 in der ersten Version veröffentlicht, haben die Common Criteria bisher weder TCSEC noch ITSEC ersetzt.² Die Common Criteria werden auch als internationale Norm ISO 15 408 veröffentlicht. Im Gegensatz zur sechsstufigen ITSEC besitzen die Common Criteria sieben Stufen der Vertrauenswürdigkeit. Der Übergang von einer Evaluierung nach TCSEC oder ITSEC zu Common Criteria ist relativ einfach, da zwischen diesen Kri-

² TCSEC, ITSEC und CC sind über viele verschiedene Server aus dem Internet kostenlos erhältlich, z. B. CC bei [NIST].

terienkatalogen viele Gemeinsamkeiten bestehen. Da jedoch gerade im Chipkartenbereich nach wie vor die ITSEC weltweit als die wesentliche Grundlage für Evaluierungen benutzt wird, wird im Folgenden nur noch auf diesen Kriterienkatalog referiert.

Verschiedentlich werden bei Evaluierungen neben ITSEC und CC noch zusätzlich die Anforderungen der FIPS 140-2 berücksichtigt. Diese Norm legt für Sicherheitsmodule, zu denen auch Chipkarten gezählt werden können, vier mögliche Sicherheitsniveaus fest und beschreibt detailliert sieben sicherheitsrelevante Anforderungsbereiche. Der Inhalt dieser Norm ist sehr praxisbezogen und geht auch auf technische Realisierungsdetails wie beispielsweise Kriterien für die Qualität von Zufallszahlengeneratoren ein.

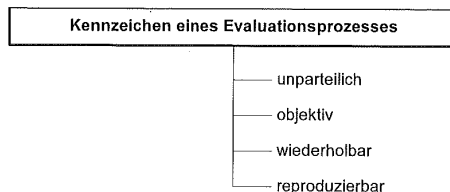


Bild 9.5 Die vier Kennzeichen eines Evaluationsprozesses.

Ein Evaluationsprozess, unabhängig von der angewandten Methodik, besitzt vier Kennzeichen: Der Prozess muss unparteiisch sein, der Prüfer darf also gegenüber dem Evaluierungsobjekt oder dem Hersteller keine Vorurteile haben. Als weiteres Charakteristikum muss der Evaluationsprozess objektiv sein und damit so strukturiert, dass persönliche Wertungen minimiert sind. Ein weiteres Merkmal besteht darin, dass man bei einer Wiederholung des Evaluationsprozesses zum selben Resultat kommen muss. Abschließend muss noch eine Reproduzierbarkeit des Evaluationsprozesses erreichbar sein, ein anderer Prüfer oder eine andere Prüfinstanz muss also zum selben Ergebnis kommen.

Einer der wichtigsten Punkte bei jeder Evaluierung ist die Festlegung der Sicherheitsvorgaben (*security target*) für das Ziel der Evaluierung (*target of evaluation – TOE*). Ziel der Evaluierung ist das zu prüfende Objekt, und die Sicherheitsvorgaben beschreiben die zu prüfenden Mechanismen. Im Übrigen kann man durch geschickte Wahl der Sicherheitsvorgaben eine Evaluierung stark vereinfachen, da sich sicherheitskritische Elemente dadurch ausgrenzen lassen. Dies ist allerdings nur ein Trick, um möglichst schnell und kostengünstig hohe Evaluationslevel zu erreichen. Die tatsächliche Sicherheit leidet dann natürlich darunter.

Die ITSEC

Da die ITSEC für alle möglichen Systeme im Bereich der Informationstechnik Gültigkeit haben soll und das Dokument nur zirka 150 Seiten hat, müssen die Sicherheitskriterien in einer sehr abstrakten Form beschrieben sein. Dies hat zur Folge, dass das Dokument sehr schwer zu lesen ist und auch mitunter – ähnlich Gesetzen – regelrecht interpretiert werden muss.

Die ITSEC geht von drei Grundbedrohungen für jedes System aus, nämlich die unbefugte Informationsgewinnung (Vertraulichkeit), die unbefugte Änderung von Informationen (Integrität) und die unbefugte Beeinträchtigung der Funktionalität (Verfügbarkeit). Auf der Grundlage dieser drei Bedrohungen sind die Sicherheitskriterien aufgebaut.

Die Bedrohungen lassen sich auch außerhalb des Systems durch verschiedene Maßnahmen reduzieren. Zum Beispiel kann ein unsicheres Computersystem dadurch erheblich an Sicherheit gewinnen, indem physikalisch der Zugang zum System etwa durch Sicherheits-schleusen reglementiert wird. Dies alles sind Rahmenbedingungen, die bei einer Beurteilung ebenfalls mit in Betracht gezogen werden müssen. Die daraus resultierende Restbedrohung kann dann durch den Anforderungskatalog der ITSEC bewertet werden.

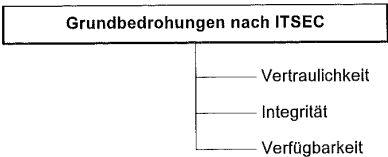


Bild 9.6 Die drei Grundbedrohungen nach ITSEC.

Der grundlegende Vorgang bei einer Evaluierung eines Systems nach ITSEC ist die Bewertung der Mechanismen zur Aufrechterhaltung der Sicherheit in Hinblick auf die drei definierten Grundbedrohungen. Die Bewertung findet dabei in schulnotenähnlichen Dimensionen statt. Die verwendeten Mechanismen müssen sowohl korrekt als auch wirksam sein. Dabei werden die Mechanismen nach sicherheitsrelevanten Grundfunktionen beurteilt, welche man auch als generische Oberbegriffe bezeichnet.

Tabelle 9.1 Die Einteilung aller Mechanismen in sicherheitsrelevante Grundfunktionen (d. h. generische Oberbegriffe) nach ITSEC.

sicherheitsrelevante Grundfunktion
Identifikation und Authentisierung
Zugriffskontrolle
Beweissicherung
Protokollauswertung
Wiederaufbereitung
Unverfälschtheit
Zuverlässigkeit der Dienstleistung
Übertragungssicherung

Die Tabelle 9.2 zeigt die Anforderungen, die für eine Evaluierung von Software für eine Chipkarte notwendig sind. Der Aufwand für die einzelnen Stufen steigt dabei jedoch nicht linear, sondern beinahe quadratisch. Um also von Stufe E2 nach E3 zu gelangen, ist der doppelte Aufwand wie von E1 nach E2 erforderlich. Dies wirkt sich naturgemäß sehr stark in den Evaluierungsstufen E4 bis E6 aus. Die vollständige Evaluierung eines mittelgroßen Chipkarten-Betriebssystems nach ITSEC E6 kann durchaus mehrere Jahre in Anspruch nehmen und mehrere Millionen Euro kosten.

Bei Evaluierungen wird noch grundsätzlich zwischen informellen, semiformalen und formalen Methoden unterschieden. Die informelle Beschreibung einer Funktion kann durchaus mit den textuellen Darstellungen in diesem Buch verglichen werden. Small-OS

aus dem Unterkapitel „5.11 Chipkarten-Betriebssystem 'Small-OS'“ ist ansatzweise semi-formal beschrieben. Eine formale Beschreibung ist hingegen logisch prüfbar und in einer formalen Notation (z. B. Prädikatenlogik) erstellt.

Tabelle 9.2 Überblick der Anforderungen an die Softwareentwicklung bei Chipkarten in Abhängigkeit der Qualitätsstufe nach ITSEC.

notwendige Information	Evaluierungsstufe					
	E1	E2	E3	E4	E5	E6
Sicherheitsvorgaben (<i>security targets</i>)	ja	ja	ja	ja	ja	ja
formales Sicherheitsmodell (<i>formal model of security policy</i>)	nein	nein	nein	ja	ja	ja
Beschreibung der Funktionen	informell	informell	informell	informell & semi-formal	informell & semi-formal	informell & formal
Beschreibung des Architekturentwurfs (<i>architectural design</i>)	informell	informell	informell	semi-formal	semi-formal	formal
Beschreibung des Feinentwurfs (<i>detailed design</i>)	nein	nein	informell	semi-formal	semi-formal	semi-formal
Implementierung (<i>implementation</i>)	nichts notwendig	nichts notwendig	nichts notwendig	Quellcodeprüfung	Quellcodeprüfung	Quell-/Objektcodeprüfung
Ausführung (<i>operation</i>)	ja	ja	ja	ja	ja	ja
Grad der Genauigkeit (<i>level of rigour</i>)	Zustandsschilderung		Darlegung		Erklärung	

Die Stufen der Absicherung gegen Bedrohungen sind in der ITSEC in mehrere Funktionalitätsklassen eingeteilt. Diese sind eine Zusammenfassung von Sicherheitsfunktionen in jeweils speziellen Kombinationen und sagen aus, welche Stufe von Bedrohungen durch das System noch sicher abgewehrt werden kann. Dabei gibt es sechs Qualitätsstufen oder Evaluierungsniveaus (E1 bis E6). Die Stufe E1 ist die niedrigste und E6 demnach die höchste Stufe der Qualität.

In allen Funktionalitätsklassen sind die formalen Anforderungen an den Entwicklungsprozess und auch an die Entwicklungsumgebung in sehr abstrakter Weise vorgeschrieben. Außerdem enthalten die Funktionalitätsklassen Informationen und Angaben über die Betriebsdokumentation sowie die spätere Betriebsumgebung. Diese Informationen sind in einer Form erstellt, die geeignet ist, alle informationstechnischen Varianten einer Software-Entwicklung abzudecken.

Bei Evaluierungen nach ITSEC wird oft noch ein Zusatz angegeben, der die so genannte „Mechanismenstärke“ betrifft. Dies ist die Robustheit gegenüber Angreifern. Es sind drei Mechanismenstärken definiert: niedrig, mittel und hoch.

„Niedrig“ kennzeichnet einen Schutz gegen zufälliges unbeabsichtigtes Eindringen in gesicherte Umgebung. „Mittel“ bedeutet, dass ein Schutz gegen Angreifer mit beschränk-

ten Mitteln vorhanden ist. Die höchste Mechanismenstärke – „hoch“ – bedeutet, dass Schutz gegen Angreifer mit sehr guten Fachkenntnissen und Mitteln vorhanden ist. Üblicherweise wird ab ITSEC E4 eine Mechanismenstärke „hoch“ verwendet. Dies sollte auch vom Standpunkt der Systemsicherheit prinzipiell bei Chipkarten gewählt werden.

Die ZKA-Kriterien

Neben den länderübergreifenden Evaluierungsstandards wie TCSEC, ITSEC oder CC haben sich bei großen Chipkarten-Anwendungen auch eigene Standards zur Evaluierung etabliert. Beispielhaft sind hier die Kriterien von Visa für Sicherheitsprüfungen und die deutschen ZKA-Kriterien. Letztere sind für die ec-Karte mit Chip in Deutschland verbindlich einzuhalten und werden von den zugelassenen Evaluierungsinstanzen für alle Chipkarten-Betriebssysteme für diese Anwendung geprüft.

Tabelle 9.3 Die 13 ZKA-Kriterien zur Evaluierung eines Chipkartensystems nach Stefan Rother [Rother 98 b] (Teil I)

Kriterium	Erklärung
Komponenten-Authentisierung	Die zum System gehörenden Sicherheitskomponenten müssen authentisierbar sein.
Nachrichtenintegrität	Die sicherheitsrelevanten Informationen zwischen den Komponenten müssen gegen Manipulationen geschützt sein.
Benutzer-Authentisierung	Bestimmte Funktionen dürfen nur dann ausgeführt werden, wenn die PIN des Benutzers – als Authentisierung des Benutzers – korrekt eingegeben wurde.
Geheimhaltung von PIN und Schlüsseln	PIN und Schlüssel dürfen außerhalb von gesicherten Bereichen nie im Klartext übertragen werden. Sofern PIN und Schlüssel in Komponenten verarbeitet oder gespeichert werden, müssen diese Komponenten gegen unautorisiertes Auslesen oder Verändern geschützt sein. Eine erschöpfende Suche nach der PIN muss systembedingt verhindert sein.
Protokollierung	Alle sicherheitsrelevanten Vorkommnisse müssen in den beteiligten Komponenten protokolliert werden. Die Protokolle müssen gegen Manipulation geschützt sein.
Schlüsselmanagement	Es muss Mechanismen zur Verteilung, Verwaltung und zum Schlüsselwechsel geben. Bei symmetrischen Kryptoverfahren dürfen nur temporäre Schlüssel verwendet werden, und die Schlüssel müssen ihrem Zweck entsprechend separiert sein.
Hardware	Alle sicherheitsrelevanten Handlungen müssen gegen unberechtigte Zugriffe geschützt sein.
Organisatorische Maßnahmen bei Herstellung und Personalisierung	Es darf nur der evaluierte Programmcode in der beschriebenen Parametrisierung im Wirkbetrieb eingesetzt werden.
Ablaufsicherung	Eine Quellcodeprüfung stellt sicher, dass die Abläufe und der Datenfluss innerhalb des evaluierten Programms korrekt sind.
andere Anwendungen	Zusätzliche Anwendungen dürfen keine sicherheitsgefährdenden Auswirkungen haben.

Tabelle 9.3 (Fortsetzung)

Kriterium	Erklärung
Verschlüsselungsverfahren	Die eingesetzten kryptografischen Algorithmen müssen dem Stand der Technik entsprechen und dürfen nicht auf der Geheimhaltung des Verfahrens beruhen.
eindeutige Repräsentation	Jede Sicherheitskomponente muss im Gesamtsystem eindeutig identifizierbar sein.
personelle Forderung	Es dürfen nur dafür geeignete Personen Zugang zu den sicherheitsrelevanten Komponenten und Mechanismen haben.

Bei der Prüfung anhand der ZKA-Kriterien werden einerseits die bereitgestellten Dokumente und andererseits die Software und auch die Hardware untersucht. Dies ist der große Vorteil der chipkartenspezifischen ZKA-Kriterien gegenüber den für jede beliebige Software gültigen TCSEC/ITSEC/CC-Kriterien. Die Tabellen 9.4 und 9.5 zeigen die 13 ZKA-Kriterien mit einer dazugehörigen Erklärung.

Zusammengefasst sind die ZKA-Kriterien wohl die zurzeit besten Sicherheitskriterien für Chipkarten, da sie die hohen Anforderungen eines großen Zahlungsverkehrssystems erfüllen und speziell auf die besonderen Belange eines Chipkarten-basierten Systems zugeschnitten sind.

Tabelle 9.4 Die Sicherheitsuntersuchungen anhand der ZKA-Kriterien an Spezifikation und im Quellcode eines Chipkarten-Betriebssystems. Nach Stefan Rother [Rother 98 a]

Spezifikation-/Quellcode-Untersuchung	
Betriebssystemkommandos	Ver- und Entschlüsselung
Anwendungskommandos	Schlüsselableitung
Kommunikationsmechanismen	Signaturerzeugung und -prüfung
Speicher- und Ressourcenverwaltung	Schlüsselverwaltung
Dateiattribute und Dateizugriffsrechte	Authentisierungsmechanismen
Prüfsummenalgorithmen	Zufallszahlengenerator

Resümee

Bei einem realen Chipkartenprojekt wird folgendermaßen vorgegangen: Der Kartenherausgeber legt mit dem Betriebssystem-Hersteller die Anforderungen für den Betrieb und die zu berücksichtigenden Bedrohungen fest. Anschließend einigt man sich auf die notwendigen Evaluationsniveaus für die Evaluierung. Danach wird das Betriebssystem entsprechend diesem Evaluationsniveau nach den entsprechenden Verfahren erstellt und mit der notwendigen Dokumentation versehen. Im abschließenden Schritt kann das fertig gestellte Betriebssystem mit all seinen Komponenten von einer entsprechenden unabhängigen Stelle, der Evaluationsinstanz, nach dem vorher vereinbarten Niveau evaluiert werden.

Eine Evaluierung nach ITSEC hat für den Kartenherausgeber und Anwendungsanbieter einige Vorteile, da er in vielen Sicherheitsaspekten Gewissheit haben kann, dass die wesentlichen Mechanismen in ihrer Konzeption und Ausführung klar definiert und funktionsfähig sind. Diese Sicherheit ist im schnelllebigen Chipkartenmarkt allerdings mit Nachteil-

len verbunden. Die Entwicklungszeit erhöht sich bei einer notwendigen Evaluierung selbst bei den niedrigen Funktionalitätsklassen erheblich. Auch verursacht der Mehraufwand bei der Dokumentation eine Steigerung der Entwicklungskosten, die Letztendlich vom Betriebssystem-Hersteller auf sein Produkt umgeschlagen wird.

Der größte Nachteil ist jedoch ganz anderer Natur. Selbst bei einem evaluierten System kann nicht garantiert werden, dass alle Abläufe und Mechanismen so funktionieren, wie sie in den entsprechenden Dokumenten beschrieben sind. Evaluierung bedeutet ja nicht, dass die evaluierende Stelle das Produkt komplett testet, sondern nur, dass diese Stelle die erhaltenen Papiere und gegebenenfalls Quell- und Objektcode zu dem Produkt prüft. Von ebensolcher Bedeutung ist, dass auch die Zielhardware eine dem jeweiligen Evaluationsniveau äquivalente Sicherheit gewährleisten muss. Eine fehlerfreie und sichere Software nützt überhaupt nichts, wenn man sie mit geringem Aufwand durch die „Hintertür“ Hardware umgehen kann. Diese vier Vorbehalte gegenüber einer Evaluierung müssen in jedem Einzelfall bedacht und auch bewertet werden.

9.3.2 Testmethoden für Software

Obwohl es unter Umständen banal klingen mag: Beim Stichwort Test sollte vorab klargestellt werden, dass das Testen eines Programms nicht in der pauschalen Suche nach Fehlern besteht, sondern in der Umsetzung einer Testspezifikation mit dem Ziel, Fehler in dem zu prüfenden Programm zu finden. Testen hat also nichts mit Debugging zu tun, da Tests von Testern durchgeführt werden und nicht von Softwareentwicklern wie beim Debugging.

Wie an anderen Stellen schon mehrmals betont, sind Programme für Chipkarten-Mikrocontroller verglichen mit anderer Software nicht sehr umfangreich. Allerdings zeichnen sie einige Eigenheiten aus. Um dies zu verdeutlichen, kann man als Beispiel einige Rahmenwerte eines typischen Betriebssystems aufführen.

Bei einem Mikrocontroller mit 16 kByte ROM und 8 kByte EEPROM benötigt die Software ca. 20 kByte Speicher, für die jeweiligen Anwendungen bleiben dann noch 4 kByte EEPROM übrig. Ist die Software in Assembler programmiert, hat sie in diesem Beispiel einen Umfang von ungefähr 30000 Zeilen Sourcecode. Druckt man dies 60-zeilig pro Seite aus, benötigt man dafür 500 Blätter Papier. Die Anzahl der bedingten Verzweigungen bewegt sich im Bereich um 2000, und selbst ein erfahrener Programmierer benötigt zur Erstellung der 20 kByte Assemblercode mindestens 9 Monate. Bei sehr leistungsfähigen Chipkarten-Betriebssystemen sind es zurzeit bis zu 120000 Zeilen C-Sourcecode, was etwa 2000 Seiten Papier entspricht.³

Dieses Zahlenbeispiel zeigt recht deutlich, dass man es bei einem Chipkarten-Betriebssystem mit einer ziemlich komplexen Software zu tun hat. Dazu kommt noch die Forderung, dass es sich dabei um eine Software handelt, die fast ausschließlich in sicherheitsrelevanten Bereichen eingesetzt wird. Es kann also nicht einfach während oder nach der Programmierung mit einigen Tests die geforderte Fehlerarmut geprüft werden. Man benötigt eine entsprechende Teststrategie.

Auch bei Chipkarten geht der Kurs weg von der Programmierung in Assembler hin zu höheren Programmiersprachen wie Hardware-nahes C oder die objektorientierte Sprache

³ Siehe auch Abschnitt 5.1 „Bisherige Entwicklung der Betriebssysteme“.

Java. Die Verwendung höherer Programmiersprachen wird bei großen Chipkarten-Betriebssystemen mit über 30 kByte Programmcode nicht nur aufgrund der Implementierungszeit notwendig, sondern auch wegen der geforderten Fehlerarmut.

Der Grund dafür lässt sich folgendermaßen erklären: Man kann davon ausgehen, dass die Anzahl der Fehler in einer Zeile Programmcode bei nahezu allen Programmiersprachen annähernd gleich ist. Da die Funktionalität bei höheren Programmiersprachen wesentlich größer als bei Assembler ist, resultiert daraus eine geringere Fehlerdichte bei gleich großem Programmcode.

Nimmt man beispielsweise den durchaus realistischen Wert von 1,5 Fehler pro 100 Zeilen Programmcode an und geht zusätzlich davon aus, dass mit vertretbarem Aufwand nur 50 % aller Fehler gefunden werden, so besitzt das getestete Programm anschließend noch immer 0,75 Fehler in 100 Zeilen Programmcode. Bei Java ist das Verhältnis von Zeilen Sourcecode zu Maschinencode (d. h. Bytecode) etwa 1 zu 6. Dies bedeutet, dass die Funktionalität einer Zeile Programmcode in Java ungefähr der von 6 Zeilen Maschinencode entspricht. Nimmt man bei sonst gleich bleibenden Randbedingungen an, dass der Kompilationsvorgang weitgehend fehlerfrei verläuft, kann man nur durch die Benutzung einer mächtigeren Programmiersprache die Anzahl der Fehler in einem Programm um den Faktor 6 senken. Selbst wenn dieser Wert in der Realität geringer ausfällt, ist es doch eine außerordentliche Motivation, leistungsstarke Programmiersprachen zu benutzen.

9.3.2.1 Grundlagen des Tests von Chipkarten-Software

Um überhaupt eine Teststrategie festlegen zu können, muss man sich zunächst einmal den Lebenszyklus von Software für Chipkarten vor Augen halten. Dazu könnte man das seit 1970 bekannte und in vielen Varianten publizierte Wasserfallmodell von W. W. Royce verwenden. Es eignet sich relativ gut für die maskenprogrammierten Chipkarten-Betriebssysteme. Da es aber auch für sehr umfangreiche Softwareprojekte für PCs oder Großrechner ausgelegt ist, sei hier im Folgenden eine vereinfachte und speziell für Chipkarten angepasste Version benutzt.⁴

Die fünf angegebenen Schritte werden normalerweise sequenziell durchlaufen. Es kann durchaus möglich sein, dass man während eines Schrittes auf ein Problem stößt, bei dem es erforderlich ist, wieder einen oder sogar mehrere Schritte zurückzugehen. Allerdings sollte dies so weit wie möglich vermieden werden, da diese Iterationen sowohl Zeit als auch Geld kosten.

Um betriebswirtschaftliche Anforderungen wie time-to-market und niedrige Durchlaufzeit in der Softwareentwicklung zu erfüllen, ist es oft notwendig, die angegebenen Schritte nicht streng nacheinander, sondern zum Teil gleichzeitig auszuführen. Mit diesem als Simultaneous Engineering bezeichneten Verfahren werden Teile der Software so früh wie möglich in einzelne Module aufgespalten und diese dann gleichzeitig durch das Wasserfallmodell geschleust. So kann es beispielsweise vorkommen, dass sich eine nur mit einem Übertragungsprotokoll ausgestattete Chipkarte schon in der Systemintegration befindet, während ein ebenfalls zur Anwendung gehöriger kryptografischer Algorithmus noch spezifiziert wird.

⁴ Eine ausführliche Beschreibung der bei der Entwicklung von Chipkarten-Betriebssystemen und Chipkarten-Anwendungen üblichen Vorgehensmodelle findet sich in Abschnitt 15.7 „Vorgehensmodelle“.

Analyse

Dieser Schritt beinhaltet die Festlegung einer grundlegenden Zieldefinition sowie die Zusammenstellung der Anforderungen zu einem Lastenheft. Dabei werden alle Anforderungen festgelegt, die von der späteren Chipkarten-Software erfüllt werden müssen. Die Analyse sieht dabei schon vor, dass erste Lösungsansätze in Form von Entwürfen beschrieben sind. Vereinfacht wird hier festgelegt, „was“ die fertig gestellte Software später zu leisten hat.

Design

In diesem dem Konzept nachfolgenden Schritt wird das „Wie-es-gemacht-wird“ festgelegt. Dazu muss eine exakte und nicht interpretierbare Spezifikation erstellt werden, die vollständig eine der möglichen Lösungen für die im Konzept geforderten Anforderungen festlegt. Ideal sind formal aufgebaute Spezifikationen, denn damit können eindeutig und unmissverständlich die Eigenschaften, Funktionen und Abläufe von Chipkarten-Software festgelegt werden. Spezifikationen in einem Pseudocode, der durch Computerprogramme auf Konsistenz und Fehlerfreiheit geprüft werden kann, eignen sich sehr gut für diese Thematik.

Im Rahmen von CASE (*computer aided software engineering*) kann dann die Spezifikation unmittelbar für die Erstellung des Sourcecodes und der Prüfprogramme weiterverwendet werden. Man spricht in diesem Zusammenhang auch oft von „ausführbaren Spezifikationen“, also Spezifikationen, die in einer von Computern interpretier- und weiterverarbeitbaren Form erstellt werden.

Realisierung und Test

Nachdem die Spezifikation abgeschlossen und abgenommen wurde, kann mit der Erstellung der Programmflusspläne für die Assembler- oder C-Programmierung begonnen werden. Daran schließt sich die Programmierung mit den dazugehörigen Tests an. Am Ende dieses Schrittes liegt ein vollständig programmiertes und ausgetestetes Chipkarten-Betriebssystem vor.

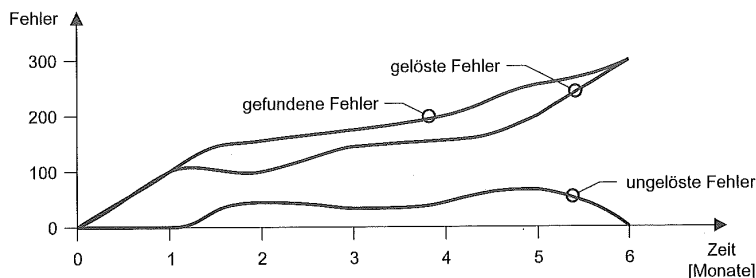


Bild 9.7 Typische zeitliche Entwicklung der gefundenen, ungelösten und gelösten Fehler im Rahmen eines typischen Entwicklungsprojekts für ein Chipkarten-Betriebssystem.

Systemintegration

Da Chipkarten nur im Zusammenhang mit einem System arbeiten können, muss in diesem Schritt die Zusammenführung der Teile vorgenommen werden. Das Ergebnis ist die vollständige und fehlerfreie Zusammenarbeit aller Teile sowie die endgültige Dokumentation des Gesamtsystems.

Wartung

Dieser letzte Schritt bei der Softwareentwicklung kann nur mehr dazu benutzt werden, um eventuell vorhandene Rahmenparameter in den ausgegebenen Chipkarten zu verändern. Größere Softwareanpassungen oder -änderungen sind in diesem Zeitraum nicht mehr möglich.

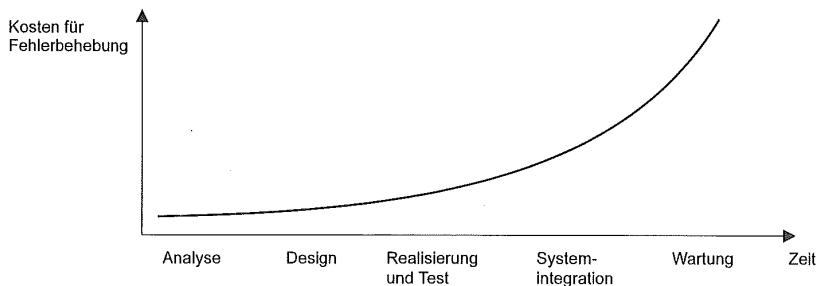


Bild 9.8 Kosten für die Behebung von Fehlern in Abhängigkeit vom Zeitpunkt der Entdeckung.

Die einfache und schnelle Programmierung von Chipkarten mittels Hochsprachen wie beispielsweise Java wird es in Zukunft ermöglichen, neben dem traditionellen Wasserfallmodell auch evolutionäre Vorgehensmodelle⁵ zu benutzen. Dabei werden die Schritte Spezifikation, Codierung, Test und zum Teil sogar Systemintegration mehrmals mit stetiger Verbesserung wiederholt. Das Ziel besteht darin, mit geringerem Spezifikationsaufwand und voll funktionsfähigen Prototypen rasch eine optimale Lösung zu schaffen.

Es ist ein seit langer Zeit bekannter Sachverhalt, dass bei Projekten aller Art und insbesondere bei der Softwareentwicklung die Fehlerbehebung umso teurer ist, je weiter man im Projekt fortgeschritten ist. Dieser Sachverhalt sollte dazu führen, dass auch in die ersten Schritte entsprechend dem Wasserfallmodell entsprechend viel Zeit und Aufwand investiert wird. Denn falls das Konzept unvollständig ist oder die Spezifikation Fehler enthält, steigen die Kosten für die Verbesserung exponentiell mit den weiteren Prozessschritten an.

9.3.2.2 Testverfahren und Teststrategien

Die Anzahl der unterschiedlichen Methoden und Testverfahren für die Softwareprüfung ist mittlerweile fast nicht mehr zu überblicken. Allerdings benötigt man nur einige wenige, doch dafür bewährte Verfahren für die Tests der Programme von Chipkarten. Man kann auf eine jahrzehntelange Erfahrung und eine große Anzahl von Publikationen beim Testen von Programmen zurückgreifen. Im Übrigen bedeutet Testen einer Software immer, dass man versucht, Fehler im Programm zu finden und nicht die Korrektheit des Programms zu beweisen.

Alle Testverfahren lassen sich in statische und dynamische einteilen. Bei den statischen Verfahren wird der Programmcode mithilfe von unterschiedlichen Methoden manuell oder automatisch analysiert und bewertet. Die zwei am häufigsten angewandten statischen Testverfahren seien hier kurz aufgeführt und erläutert:

⁵ Siehe auch Abschnitt 15.7 „Vorgehensmodelle“.

Statistische Programmauswertung mit Softwarewerkzeugen

Dies ist die Analyse diverser Eigenschaften des Programmcodes, mit statistischen Verfahren analysiert, beispielsweise:

- Anzahl der Zeilen Programmcode (*lines of code – LOC*)
- Anzahl der Zeilen Kommentare (*lines of comment*)
- Verhältnis des Umfangs der Kommentare zum Programmcode
- Aufteilung des Programmcodes
- Anzahl der Funktionen
- Verschachtelungstiefe
- toter Code

Review

Dies ist die formale Analyse und Bewertung der Programmmodule durch ein Team von Gutachtern. Sie wird in Spezialfällen auch als Code-Walk-Through oder Code-Inspektion bezeichnet. Grundsätzlich wird dabei geprüft, ob Codierung, Struktur und Datenfluss mit der Spezifikation übereinstimmen.

Im Gegensatz zu den statischen Verfahren wird bei der dynamischen Programmanalyse das zu prüfende Programm während des Betriebs von Hand oder unter Zuhilfenahme von Computern getestet. Dabei gibt es zwei grundsätzlich unterschiedliche Ansätze und als dritte Möglichkeit eine Mischform:

Blackbox-Test

Beim Blackbox-Test geht man davon aus, dass der testenden Instanz nichts über die internen Abläufe, Funktionen und Mechanismen des zu testenden Programms bekannt ist. Dies führt dazu, dass man nur die Eingabe- und Ausgabedaten mit den in der Spezifikation festgelegten Beziehungen zueinander betrachtet.

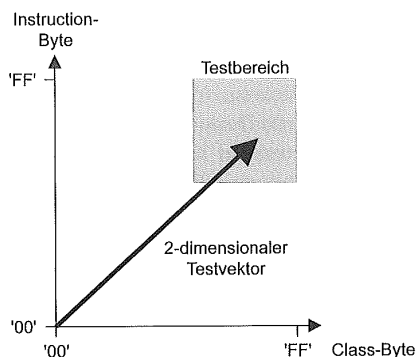


Bild 9.9 Beispiel für einen 2-dimensionalen Testvektor beim Blackbox-Test, der einen über Äquivalenzklassen festgelegten Testbereich bestimmt. Bei komplexen Testfällen kann der Testvektor durchaus 10 und mehr Dimensionen erreichen. Aufgrund der limitierten Testzeit muss man deshalb den oder die Testbereiche entsprechend begrenzen.

Blackbox-Tests sind bei Chipkarten-Betriebssystemen der Standard. Sie werden auch für Sicherheitsmodule von Terminals oder Computersystemen benutzt. Oft wird aber fälschlicherweise angenommen, dass mit diesen Tests neben Fehlern auch eventuell vorhandene trojanische Pferde oder Ähnliches entdeckt werden können. Mit dieser Annahme möchte man sich dann eine relativ aufwendige und teure Analyse des Programmcodes ersparen. Es mag möglich sein, einfache und unüberlegt programmierte oder unabsichtlich erstellte Hintertüren in Chipkarten-Betriebssystemen damit zu erkennen, doch kann ein erfahrener Programmierer ohne weiteres Zugangsmöglichkeiten schaffen, die mit einem Blackbox-Test nie aufzuspüren sind. Ein kleines Beispiel mag dies verdeutlichen. Es soll nicht als Vorbild für trojanische Pferde dienen, da es im Übrigen schon seit langem bekannt ist, sondern vor allem dazu da sein, um das Bewusstsein über die Notwendigkeit von Code-Inspektionen für Sicherheitsanalysen zu schärfen.

Man könnte das in fast jedem Chipkarten-Betriebssystem vorhandene Kommando für die Erzeugung und Ausgabe von Zufallszahlen (GET CHALLENGE) dahingehend abändern, dass nur die erste ausgegebene 8 Byte lange Zufallszahl durch den Pseudozufallszahlengenerator erzeugt wird. Die anschließenden „Zufallszahlen“ sind jeweils 8 Byte EEPROM-Speicherinhalt, mit der ersten Zufallszahl durch XOR verknüpft. Mit einem einfachen Programm kann dann von außen der gesamte Speicher inklusive aller Schlüssel ausgelesen werden. Dies ist im Übrigen ein sehr gutes Beispiel für angewandte Steganografie bei Chipkarten.

Es besteht keine Möglichkeit, mit einem Blackbox-Test festzustellen, ob sich hinter diesem Kommando ein trojanisches Pferd versteckt. Selbst mit einer statistischen Analyse der erhaltenen Zufallszahlen würde man hier keine signifikante Abweichung von den üblichen Pseudozufallszahlen feststellen. Die einzige Art, dieses manipulierte Programm zu erkennen, ist eine Inspektion des gesamten Programmcodes des Betriebssystems. Das Beispiel zeigt nur eine von vielen denkbaren Varianten, um ein übliches Chipkarten-Kommando dahingehend abzuwandeln, dass man Dateninhalte des Speichers erhält. Da der dafür notwendige Programmcode nur einige Zeilen umfasst, ist die einzig wirksame Methode, um diese Möglichkeiten auszuschließen, eine vollständige Offenlegung und Analyse des Sourcecodes.

Zur Prüfung der Funktionsfähigkeit von atomaren Abläufen werden Abbruchtests durchgeführt. Diese nennt man auch Recovery Tests. Dazu sendet man ein entsprechendes Kommando zur Chipkarte, in der daraufhin ein atomarer Ablauf gestartet wird. Während der Zeitdauer des atomaren Ablaufs wird die Spannungsversorgung zu einem bestimmten Zeitpunkt unterbrochen. Anschließend prüft man, ob die verarbeiteten Daten durch den atomaren Zugriff in einem konsistenten Zustand gehalten wurden. Die Spannungsabschaltung führt man nicht nur zu einem bestimmten Zeitpunkt aus, sondern führt diese Tests in dem gesamten Bereich des atomaren Ablaufs wiederholt aus. Um korrekte Testergebnisse zu erhalten, versetzt man den Abbruchzeitpunkt schrittweise um etwa jeweils die halbe EEPROM-Schreibzeit für eine Seite. Damit lässt sich sehr gut die Funktionsweise atomarer Abläufe prüfen. Die Anzahl der Tests dafür ist allerdings nicht unerheblich. Um einen Zeitraum von 100 ms für die Ausführung eines Kommandos in der Chipkarte mit dieser Methode abzutesten, sind bei einer typischen zeitlichen Schrittweite von 1 ms 100 Tests notwendig.

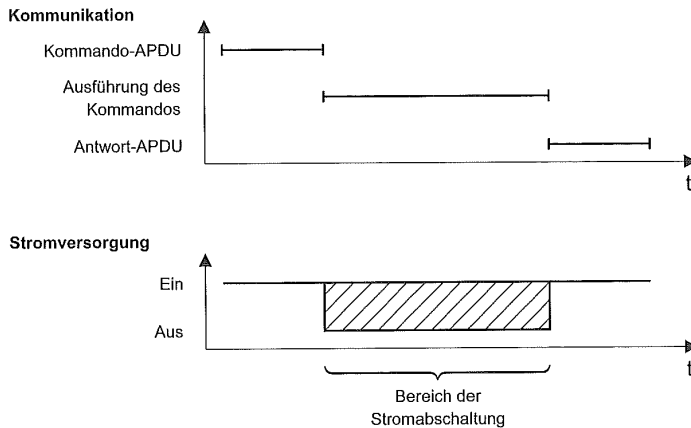


Bild 9.10 Zeitlicher Ablauf bei Abbruchtests zur Prüfung atomarer Abläufe.

Whitebox-Test

Der Whitebox-Test wird häufig auch Glassbox Test genannt, was recht eindeutig die dahinterstehende Philosophie zeigt. Alle internen Datenstrukturen und Abläufe sind bekannt und können vollständig nachvollzogen werden. Zur Konzeption und Erstellung der Tests dienen die entsprechenden Programmdokumentationen, doch ist die Spezifikation immer die alleinige Referenz. Eine seit Jahrzehnten übliche Form der Dokumentation sind Programmflusspläne oder auch Nassi-Sneidemann-Diagramme (Struktogramme), welche auch beim Whitebox-Test die Grundlage für die Beurteilung der internen Funktion der Software sind. Bei objektorientierten Programmiersprachen wie Java hat sich UML (*unified modeling language*) als die übliche Darstellungsform durchgesetzt. Die verschiedenen Beschreibungsformen der UML eignen sich ebenfalls sehr gut zur Architekturbeschreibung von Chipkarten-Software.

Da man damit die genauen Programmläufe kennt, möchte man natürlich alle möglichen Ablaufpfade im Programm prüfen. Dabei gibt es mehrere Vorgehensweisen: Bei der Befehlsabdeckung (*statement coverage*) wird jeder Befehl im Programm mindestens einmal durchlaufen. Man kann mit dieser Analyse beispielsweise sehr einfach herausfinden, ob im Programm toter Code enthalten ist, der nie benutzt wird. Für eine Sicherstellung der gewünschten Funktion ist dieser Test zu schwach. Besser ist es, die Entscheidungsabdeckung (*decision coverage*) zu verwenden, weil dabei alle vorhandenen Entscheidungen im Programmcode mindestens einmal in allen möglichen Varianten durchlaufen werden müssen.

Um diese internen Programmläufe beim dynamischen Test erkennen zu können, bedarf es entweder eines hoch entwickelten Emulators für den jeweiligen Chipkarten-Mikrocontroller, oder das zu prüfende Programm muss instrumentiert werden. Dies bedeutet, dass vor allen Sprüngen, Verzweigungen und Funktionsaufrufen spezieller Programmcode eingefügt wird, der dann beim Durchlaufen Informationen über Ort und Parameter abspeichert. Mit einem Analyseprogramm lassen sich dann diese Informationen statistisch und grafisch auswerten. Leider führt der zusätzliche Programmcode der Instrumentierung zu einem anderen Zeitverhalten und kann im ungünstigsten Fall sogar ein unterschiedliches Verhalten des Programms auslösen. Dies muss bei dieser Analysemethode bedacht werden.

Die Erweiterung des Kriteriums der Entscheidungsabdeckung besteht darin, dass alle Entscheidungen im Programm in jeder möglichen Kombination einmal durchlaufen werden. Damit sind alle möglichen Ablaufpfade abgedeckt. Dies ist aber aufgrund der eingeschränkten Testzeit nur bei sehr kleinen Programmen mit einigen hundert Byte Codeumfang möglich. Selbst bei Programmen in der Größenordnung von 1000 Byte kann ein Test aller Kombinationsmöglichkeiten nicht mehr innerhalb einer vernünftigen Zeitspanne durchgeführt werden.

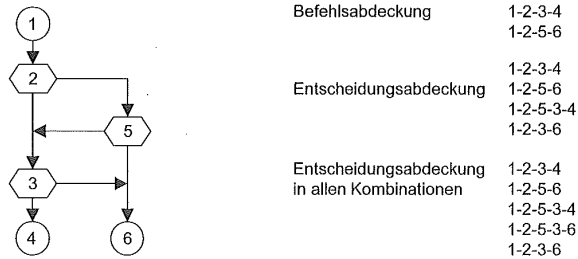


Bild 9.11 Beispiel für die Anzahl der möglichen Ablaufpfade in einem Programmflussplan bei Befehls- und Entscheidungsabdeckung.

Tabelle 9.6 zeigt dazu einen kurzen Überblick anhand eines typischen Kommandointerpreters für Chipkarten. Die Funktion dieses Programmmoduls besteht darin, ein im Empfangspuffer der Chipkarte abgelegtes Kommando anhand von Class und Instruction Byte zu erkennen und anschließend eine Prüfung von P1, P2, Lc und Le vorzunehmen. Dieses Programm hat einen Codeumfang von ca. 200 Byte und besitzt 18 Verzweigungen im Programmcode. Die möglichen Ausgabewerte sind 5 Returncodes sowie der Aufruf von 26 verschiedenen Commandos.

Gerade beim Test von Chipkarten-Betriebssystemen benutzt man oft noch zwei weitere Kriterien zur Pfadabdeckung. Es ist dies die Eingabe- und die Ausgabeüberdeckung. Das Ziel ist, dabei alle möglichen Werte für die Eingabe bzw. die Ausgabe zu erzeugen. Oft reduziert man die möglichen Werte für die Ausgabe auf die vorhandenen Returncodes, da sonst die Anzahl der Varianten zu groß wäre.

Da die Zahl der möglichen Eingabewerte ebenfalls sehr schnell Größenordnungen erreicht, die aufgrund ihrer Vielzahl bzw. des dafür notwendigen Zeitaufwands nicht mehr testbar sind, benutzt man üblicherweise so genannte Äquivalenzklassen. Damit reduziert man die sehr hohe Anzahl an möglichen Eingabewerten auf einige wenige, die dann innerhalb einer überschaubaren Zeitspanne getestet werden können. Man bildet Äquivalenzklassen, indem man auf die Grenzfälle beiderseits der testenden Entscheidung prüft und zusätzlich noch einen typischen Wert in der Mitte des möglichen Bereichs verwendet.

Erlaubt beispielsweise der Kommandointerpreter in der Chipkarte für das P1-Byte den Bereich von einschließlich 20 bis einschließlich 50, würde man die folgenden Werte als Äquivalenzklasse verwenden: 19, 20, 50, 51 und als Wert in der Mitte beispielsweise 35. Damit hätte man dann die wesentlichen Abfragebedingungen im Programm überprüft. Mit relativ großer Wahrscheinlichkeit kann man nach diesem Test davon ausgehen, dass die Abfrage des Wertebereichs richtig implementiert wurde.

Es ist vor allem in der Assemblerprogrammierung leider notwendig, bei der Festlegung der Äquivalenzklassen die Eigenschaften der Zielhardware zu berücksichtigen. So müssen

beispielsweise alle mathematischen Operationen, die aufgrund der Rechnerarchitektur (8/16/32 Bit) einen Über- oder Unterlauf im Prozessor verursachen, bei der Äquivalenzklassenbildung berücksichtigt werden. Nur so kann man sicherstellen, dass Über- bzw. Unterträge bei der Programmierung korrekt berücksichtigt wurden.

Whitebox Tests finden sich bei Chipkarten-Entwicklungen oft bei Modultests. Dabei werden fertig entwickelte Software-Module in der Chipkarte von außen über spezielle Testkommandos mit Daten beaufschlagt und anschließend die Arbeitsergebnisse der Software-Module von außen durch Testkommandos ermittelt. Außerhalb der Chipkarte findet dann der Vergleich des erhaltenen mit dem erwarteten Ergebnis statt.

Tabelle 9.5 Gegenüberstellung der Anzahl der möglichen Testfälle in Abhängigkeit von verschiedenen Verfahren zur Abdeckung beim Whitebox Test. Als Beispiel wurde ein 200 Byte großer Kommandointerpreter für Chipkarten benutzt. Für die Berechnung der Testzeit wurde eine durchschnittliche Abarbeitungszeit inklusive Datenübertragung von 30 ms angenommen.

Abdeckungsverfahren	Anzahl der möglichen Testfälle	Dauer der Tests
1 Mio. zufällige Eingabewerte	1 000 000	≈ 8 h
Befehlsabdeckung	10	≈ 0,3 s
Entscheidungsabdeckung	50	≈ 1,5 s
Entscheidungsabdeckung in allen möglichen Variationen	50 000 000	≈ 17 Tage
Eingabeüberdeckung in allen möglichen Variationen (5 Byte Header)	≈ 1,1 · 10 ¹¹	≈ 1 000 Jahre
Eingabeüberdeckung mit Äquivalenzklassen	15	≈ 0,5 s
Ausgabeüberdeckung der Returncodes	6	≈ 0,2 s

Greybox-Test

Eine Mischform zwischen Blackbox und Whitebox Test stellt der Greybox-Test dar. Bei ihm sind Teile der Software bekannt, wie beispielsweise die internen Programmabläufe. Er wird im Bereich der Chipkarten vor allem in der Integrationsphase eingesetzt, da man Fehler bei der Zusammenarbeit der einzelnen Komponenten sehr schnell und effektiv erkennen und beseitigen kann. Dazu benötigt man seitens des Schlüsselmanagements natürlich entsprechende Testschlüssel (*testkey*), die öffentlich sind. Ist dieser Teil des Integrationstests erfolgreich abgeschlossen, kann man die Ergebnisse mit den Echtschlüsseln (*lifekey*) überprüfen.

9.3.3 Dynamische Tests von Betriebssystemen und Anwendungen

Es ist wichtig, dass man sich von Anfang an klarmacht, dass durch den Test von Programmen lediglich die Anwesenheit von Fehlern bewiesen werden kann, nicht deren Abwesenheit. Geht man davon aus, dass nur ungefähr die Hälfte aller vorhandenen Fehler überhaupt gefunden wird, so besitzt ein durchschnittliches Programm sicherlich noch einige Schwachstellen.

In der Praxis wird bei der Assemblerprogrammierung oft von einer Rate von 0,7 Fehler pro 100 Zeilen Code nach allen Tests ausgegangen. Benutzt man obige Angabe von 30000

Zeilen Programmcode für ein Chipkarten-Betriebssystem als Anhaltswert und reduziert 2/3 aller Zeilen als Kommentar, kann man sich ausrechnen, dass in einem vollständig ausgetesteten und freigegebenen Betriebssystem noch etwa 70 Fehler vorhanden sind. In den sicherheitskritischen Bereichen der Militär- oder Medizintechnik wird trotz des immensen Aufwands für Test- und Qualitätssicherung immerhin noch von 4 nicht gefundenen Fehlern bei 10000 Zeilen Programmcode ausgegangen [Thaller 93]. Die meisten Fehler treten zwar nie zu Tage, doch genügt unter Umständen ein einziger, um alle Sicherheitsbarrieren eines Chipkarten-Betriebssystems zu umgehen. Es ist sehr nützlich, sich diese Aussage als Motivation für sorgfältige und überlegte Tests immer vor Augen zu halten.

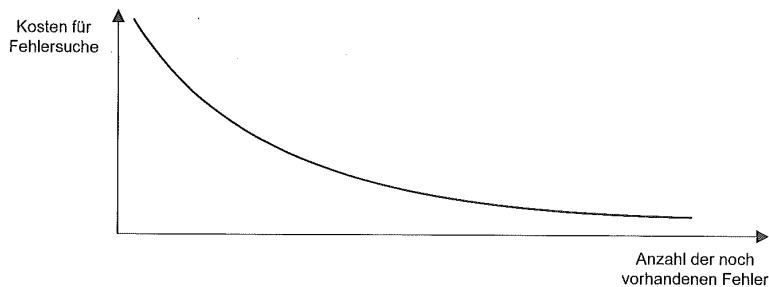


Bild 9.12 Kosten für die Fehlersuche in Abhängigkeit von noch vorhandenen Fehlern.

Selbstverständlich sind auch den Tests natürliche Grenzen gesetzt. Im Gegensatz zu Forschungsprojekten sind gerade bei Kundenprojekten die zur Verfügung stehende Zeit oder die maximal aufwendbaren Kosten ein stark einschränkendes Kriterium. Außerdem werden die Tests immer schwieriger und aufwendiger, wenn das Programm nur mehr wenige Fehler enthält. So hat irgendwann die Suche nach den letzten Fehlern in einem Programm ein Ende, da die dafür aufgewendete Zeit und die Ressourcen grundsätzlich begrenzt sind.

Bei der Ausgabe neuer Versionen kann man zunächst davon ausgehen, dass diese immer weniger Fehler enthalten, da man die Möglichkeit hat, die während des Betriebs gefundenen Fehler zu analysieren und dann zu eliminieren. Interessanterweise setzt sich der Rückgang der Fehlerzahl aber nicht unbegrenzt fort, sondern man kann nach einem Minimum im Bereich der zweiten Version im Allgemeinen wieder einen Anstieg beobachten. Dies rührt einfach daher, dass die notwendigen Fehlerbehebungen auf der ursprünglichen Spezifikation und dem Programmcode aufbauen. Ab einem bestimmten Zeitpunkt, der variabel sein kann, ist es wahrscheinlich, dass durch die Beseitigung eines Fehlers wieder ein oder sogar mehrere neue Fehler entstehen. Dies führt zu der in Bild 9.13 dargestellten Kurve. Es ist also nach einer bestimmten Anzahl von Versionen wesentlich besser, einen vollständigen Neuanfang zu machen, als auf überholten Konzepten und mehrfach ausgebessertem Programmcode aufzubauen. Im Übrigen gilt dies für fast alle Bereiche der Technik.

Nach der Norm Standard for Software Unit Testing IEEE 1008 unterscheidet man bei dynamischen Tests drei Testniveaus. Beim Basic Test werden im Wesentlichen der Gutfall und die grundlegenden Funktionen der einzelnen Kommandos geprüft. Der Capability Test umfasst bereits Grenzwerte und Schlechtfälle. Das dritte Testniveau ist der Behavior Test, bei dem die Kommandos im Zusammenspiel mit anderen geprüft werden.

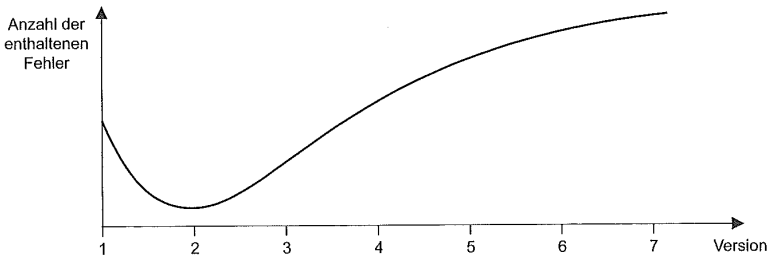


Bild 9.13 Empirisch ermittelte Abhängigkeit der in einem Programm noch vorhandenen Fehler in Abhängigkeit von der Version.

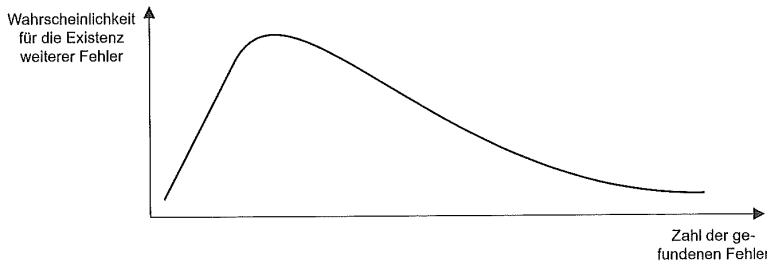


Bild 9.14 Empirisch ermittelte Abhängigkeit der in einem Programm noch vorhandenen Fehler in Abhängigkeit von den gefundenen Fehlern.

Vorgehensweise bei Tests

Es ist ein großer Unterschied, ob ein neues Betriebssystem oder eine neue Anwendung getestet werden muss. Beim Test eines Chipkarten-Betriebssystems muss der gesamte Programmcode für die unterschiedlichsten Anwendungsfälle geprüft werden. Dies erfordert eine große Anzahl unterschiedlicher Tests. Bei einer neuen Anwendung, die bekanntermaßen nur aus einem DF und mehreren EFs besteht, reduzieren sich die Tests entsprechend den eingebrachten Daten und den dafür definierten Identifizierungs- und Authentisierungsverfahren.

Muss ein neues Betriebssystem getestet werden, so erstellt man in der Regel mehrere Testanwendungen, die einige typische spätere Anwendungen wiedergeben. Man bildet sozusagen Äquivalenzklassen für die üblichen Anwendungen. Auf dieser Grundlage können dann die einzelnen Tests durchgeführt werden.

Die folgende Vorgehensweise hat sich beim Testen von neuen Betriebssystemen für Chipkarten seit einigen Jahren und bei den unterschiedlichsten Projekten eingebürgert: Es wird immer mit dem Test der Datenübertragung begonnen, da dies die Basis für alle weiteren Aktionen darstellt. Anschließend werden alle verfügbaren Kommandos geprüft. Falls es sich um eine Anwendung handelt, schließt sich daraufhin der Test der Dateien an. Falls alle vorgenannten Tests positiv verlaufen sind, kann man mit der Prüfung der festgelegten Abläufe beginnen.

Es gibt momentan auf internationaler Ebene wenige Normen, die den Aufbau und die Durchführung von Tests für Chipkarten-Betriebssysteme oder Chipkarten-Anwendungen regelt. Die europäische Norm (EN 1292) definiert einige Tests für den ATR und das Übertragungsprotokoll T = 1. Für die GSM-Chipkarten sind in der GSM 11.17 verhältnismäßig umfangreiche Tests für Betriebssystem und Anwendung festgelegt.

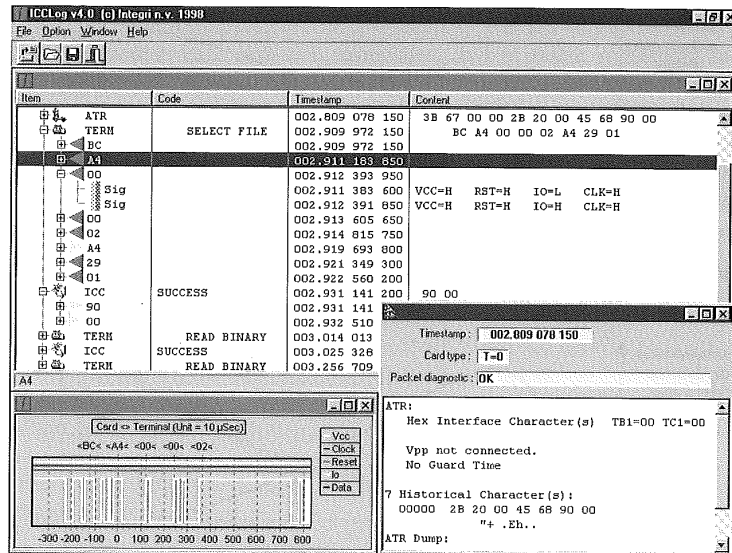


Bild 9.15 Bildschirmausgabe eines Werkzeuges zur Überprüfung der Kommunikation zwischen Terminal und Chipkarte auf physikalischer und logischer Ebene. (Copyright by Integri [Integri])

Um einen Überblick zu geben, ist deshalb in den folgenden Zeilen eine Sammlung möglicher Tests in einer üblichen Reihenfolge aufgeführt. Die Liste erhebt keinen Anspruch auf Vollständigkeit und ist als detailliertes Anschauungsbeispiel gedacht. Der Zweck der genannten Tests besteht darin, ein neues Betriebssystem inklusive einer oder mehrerer Anwendungen auf die wesentlichen Rahmenwerte zu prüfen.

Test der Datenübertragung

- ATR (Erkennung von Paritätsfehlern; falls T = 0 vorhanden, dann Zeichenwiederholung, Struktur und Inhalt des ATR)
- PTS (Struktur und Inhalt des PTS)
- Test der Datenübertragung auf OSI-Schicht 2 (Start-, Daten-, Stopbits, Teiler, Convention der Übertragung)
- Test des Übertragungsprotokolls T = 0 (Erkennung von Paritätsfehlern und Zeichenwiederholung, diverse Abläufe)
- Test des Übertragungsprotokolls T = 1 (CWT, BWT, BGT, Resynch, Fehlermechanismen, diverse Abläufe)
- Secure Messaging

Test der vorhandenen Kommandos

- Test aller möglichen Class Bytes
- Test aller möglichen Instruction Bytes
- Test aller vorhandenen Kommandos mit Äquivalenzklassen der unterstützten Funktionalitäten

Test der vorhandenen Dateien

- Test, ob alle Dateien an der richtigen Position (MF, DF) vorhanden sind
- Test auf korrekte Dateigröße
- Test auf korrekte Dateistruktur
- Test auf korrekte Dateiattribute
- Test auf die korrekten Inhalte der Dateien
- Test der festgelegten Zugriffsbedingungen (Lesen, Schreiben, Sperren, Entsperren, ...)

Test der vorhandenen Abläufe

- Test der festgelegten Zustandsautomaten (d. h. Reihenfolge der Kommandos)

Wie man sich unschwer vorstellen kann, benötigt man trotz Äquivalenzklassenbildung und einigen weiteren Minimierungsverfahren eine relativ große Menge einzelner Tests. Dabei ist davon auszugehen, dass man für ein 20 kByte großes Chipkarten-Betriebssystem zwischen 4 000 und 8 000 unterschiedliche Tests festlegen muss, um die wesentlichen Testfälle abzudecken. Tests, die beispielsweise in einer Schleife mehrere hundert verschiedene Werte zur Chipkarte senden, werden als ein Test gezählt. Die Anzahl der an die Chipkarte

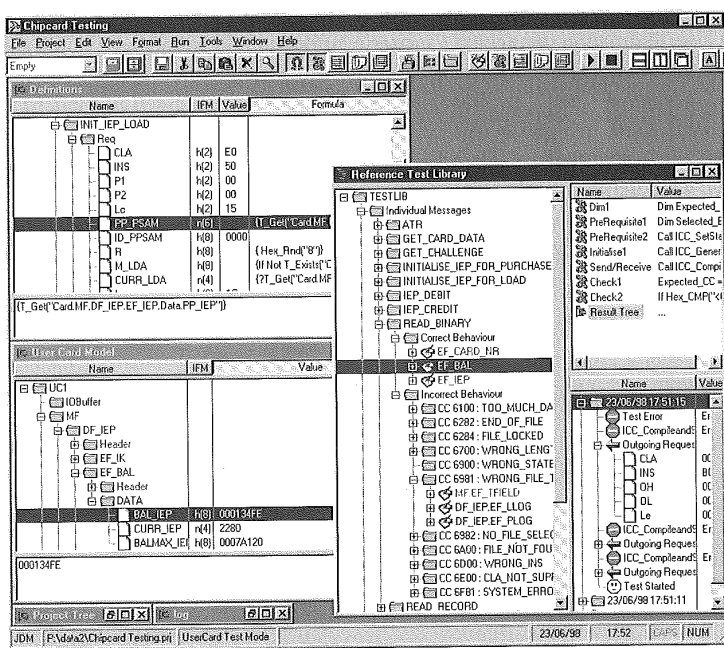


Bild 9.16 Bildschirmausgabe eines objektorientierten und datenbankgestützten Testwerkzeugs für die Prüfung von Chipkarten-Betriebssystemen und Chipkarten-Anwendungen. Das Bild zeigt oben links die Definition der Datenelemente einer Kommando-APDU (INITIALIZE IEP for LOAD) und darunter die dazugehörige Referenzsimulation der Chipkarte. Auf der rechten Seite ist ein Ausschnitt der Baumstruktur für die Festlegung der einzelnen Tests dargestellt. (Copyright by Integri [Integri])

gesendeten Kommandos kann sich dann durchaus in einer Größenordnung von 40 000 bewegen. Die Dauer für die Ausführung aller Tests liegt im Bereich von ein bis zwei Tagen. Diese große Anzahl von Tests kann man im Rahmen eines akzeptierbaren Aufwands nur mehr mit Hilfe einer geeigneten Datenbank verwalten. Diese kann dann gleichzeitig auch die Ergebnisse der Tests aufnehmen.

Für die formale Beschreibung der Tests kann beispielsweise die in ISO/IEC 9646-3 normierte TTCN (*the tree and tabular combined notation*) verwendet werden. Damit lassen sich in allgemeiner und normierter Form beliebige Testfälle definieren, aus denen anschließend automatisch ein Interpreter die Kommando-APDUs für die zu testende Chipkarte generieren kann. Nun können weitgehend automatisierte Testabläufe definiert werden.

Der Aufbau eines Testwerkzeugs für Chipkarten ist in Bild 9.15 dargestellt. Die Spezifikation für die Software in der Chipkarte ist vollständig in einem Pseudocode geschrieben worden und befindet sich in einer entsprechenden Datenbank. Ändert sich die Spezifikation, führt dies automatisch zu den notwendigen Änderungen bei den Tests. Eine weitere Datenbank enthält alle Tests in einer ebenfalls direkt durch einen Computer auswertbaren Hochsprache. Die beiden Datenbanken speisen einen Testmuster-Generator, der die Kommandos (d. h. TPDU's bzw. APDU's) für die zu testende Chipkarte generiert. Parallel dazu wird eine Simulation der echten Chipkarte betrieben, die weitgehend durch die Spezifikation definiert ist. Da in der echten Chipkarte Prozesse ablaufen können, die nicht vollständig vorhersagbar sind (z. B. Erzeugung einer Zufallszahl), müssen zusätzliche Daten an die simulierte Karte übergeben werden. Sowohl die echte als auch die simulierte Chipkarte senden die Antworten auf die erhaltenen Kommandos an einen Vergleich. Sind beide gleich, dann liefert die echte Chipkarte das richtige Ergebnis, sofern die Simulation als Referenz anerkannt ist. Alle bei einem Testlauf entstehenden Daten werden in einer Protokolldatenbank abgelegt, wo sie dann zeitlich versetzt manuell ausgewertet werden.

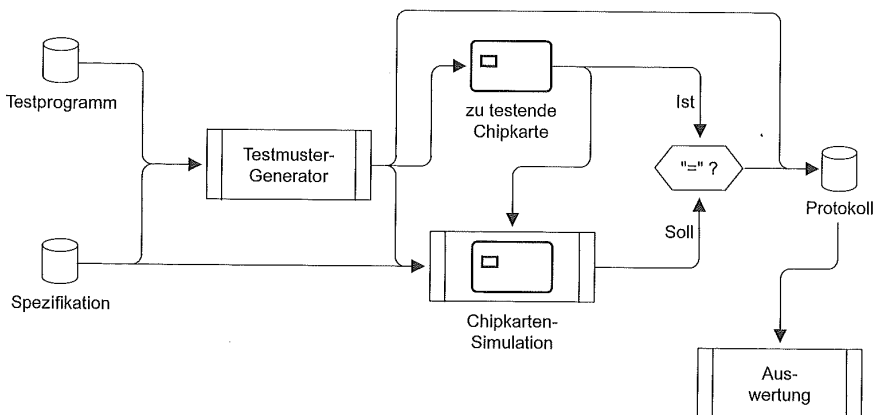


Bild 9.17 Prinzipieller Aufbau eines Testwerkzeugs für die Prüfung von Chipkarten-Betriebssystemen und Anwendungen.

10 Lebenszyklus einer Chipkarte

10.1	Die 5 Phasen des Chipkarten-Lebenszyklus	608
10.2	Phase 1 des Lebenszyklus im Detail	610
10.2.1	Erstellung des Betriebssystems und Herstellung der Chips	610
10.2.2	Herstellung der Kartenkörper ohne integrierte Spule	622
10.2.3	Herstellung von Kartenkörpern mit integrierter Spule	630
10.2.4	Zusammenführen von Kartenkörper und Chip	635
10.3	Phase 2 des Lebenszyklus im Detail	638
10.4	Phase 3 des Lebenszyklus im Detail	644
10.5	Phase 4 des Lebenszyklus im Detail	656
10.6	Phase 5 des Lebenszyklus im Detail	658

Dieses Kapitel zeigt den Werdegang einer Chipkarte von der Entstehung des Halbleiters über die Kartenherstellung bis zum Recycling der Karte. Dem Lebenszyklus von Chipkarten-Anwendungen ist hier zusätzlich ein eigener Abschnitt gewidmet, da dies bei den immer zahlreicheren Multiapplikationskarten von großer Bedeutung ist. Zu den wichtigsten Lebensabschnitten einer Chipkarte (wie etwa die Grundlagen von Chipkarten-Betriebssystemen) existieren ohnehin eigenständige Kapitel. An den betreffenden Stellen wird darauf explizit verwiesen. Zu Materialien für den Kartenkörper, zu unterschiedlichen Modultypen und Kartenelementen sei gleich an dieser Stelle auf die entsprechenden Abschnitte in Kapitel 3 verwiesen.¹

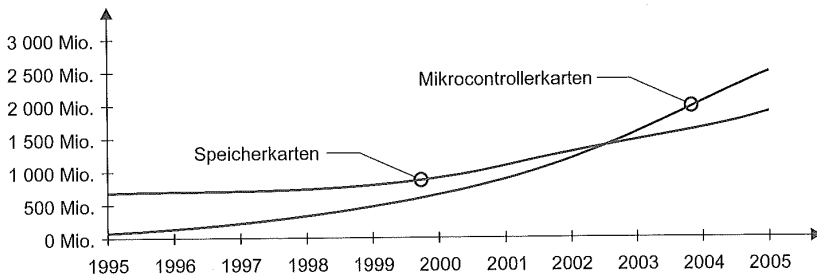


Bild 10.1 Darstellung der weltweiten Produktion von Speicher- und Mikrocontrollerkarten. Die Zahlen sind Schätzwerte, da die unterschiedlichen Quellen erheblich differieren und sie deshalb als Mittelwert gerechnet wurden.

Eine Chipkarte setzt sich prinzipiell aus zwei völlig unterschiedlichen Komponenten zusammen. Die erste Komponente ist der Kartenkörper mit Aufdruck, Sicherheitsmerkmalen und einer eventuell vorhandenen Magnetpiste. Die zweite Komponente, die den Kartenkörper schließlich zur vollwertigen Chipkarte macht, ist das Modul mit dem darin enthaltenen Chip. Diese Einteilung ist noch unabhängig davon, ob es sich um Karten mit Speicherchips oder Mikrocontrollern handelt.

¹ Siehe auch Abschnitt 3.2 „Kartenkörper“.

Auch die Art und Weise der Datenübertragung hat Auswirkungen auf den Kartenaufbau. Bei kontaktbehafteten Chipkarten wird die elektrische Verbindung zum Terminal über die außen sichtbaren sechs oder acht Kontaktflächen hergestellt. Die kontaktlosen Chipkarten enthalten im Kartenkörper eine mehr oder minder große Spule, die an dem ebenfalls im Kartenkörper eingebetteten Chip angeschlossen ist. Dieser Aufbau beeinflusst natürlich in erheblicher Weise die Herstellung der jeweiligen Kartenart.

Der Fertigungsprozess hängt auch in erheblicher Weise von den Kartenelementen ab, wie Material des Kartenkörpers, Beschriftungsarten oder Sicherheitsmerkmale. Doch unabhängig von allen diesen Varianten hat vor allem eines absolute Priorität: Kostenoptimierung. Die Fabrikation von Chipkarten ist eine Massenfertigung, bei der die Losgrößen bei etwa 10000 Stück beginnen, aber durchaus auch den 10 Mio.-Bereich erreichen können. Eine hochoptimierte Fertigung ist die wichtigste Voraussetzung für kostengünstige und qualitativ hochwertige Kartenprodukte.

10.1 Die 5 Phasen des Chipkarten-Lebenszyklus

Der Lebenszyklus einer Chipkarte ist neben dem Fertigungsprozess davon abhängig, in welcher Anwendung die Karte eingesetzt wird. Eine Chipkarte für das GSM-Mobilfunksystem hat beispielsweise nach der Herstellung einen erheblich anderen Weg vor sich als eine Kreditkarte mit Chip, doch gibt es auch sehr viele Gemeinsamkeiten unter den verschiedenen Karten.

Tabelle 10.1 Die einzelnen Phasen nach ISO 10 202-1 mit den dazugehörigen Aktionen im Überblick

Phasen des Kartenlebenszyklus	typische Aktionen
Phase 1 – Herstellung von Chip und Chipkarte	• Design der Chips • Erstellung des Chipkarten-Betriebssystems • Produktion der Chips und Module • Herstellung des Kartenkörpers • Einbetten der Module in den Kartenkörper
Phase 2 – Kartenvorbereitung	• Chipkarten-Betriebssystem komplettieren
Phase 3 – Anwendungsvorbereitung	• Initialisierung der Anwendung(en) • Personalisierung (d. h. Individualisierung) der Anwendung(en) (optisch und elektrisch)
Phase 4 – Kartenbenutzung	• Aktivierung von Anwendungen • Deaktivierung von Anwendungen
Phase 5 – Ende der Kartenbenutzung	• Deaktivierung der Anwendungen • Deaktivierung der Karte

Die ISO 10 202-1 versucht einen Kartenlebenszyklus (*card life cycle*) zu definieren, der sowohl alle Herstellungsverfahren als auch die unterschiedlichsten Anwendungen gleichermaßen abdeckt. Diese Norm ist sehr stark auf Zahlungsverkehrsanwendungen und die dortige Informationstechnik ausgerichtet und weniger auf die eigentliche Herstellung von Kartenkörper und Chip. Sie stellt jedoch einen recht gut gelungenen Versuch dar, den Werdegang von Chipkarten vom Anfang bis zum Ende strukturiert aufzuzeigen. Deshalb wird sie hier auch im Folgenden als Basis zur Beschreibung des Kartenlebenszyklus benutzt.

Eingeteilt ist das „Leben“ einer Karte nach ISO 10 202-1 in fünf Phasen, die durch genau festgelegte Übergänge miteinander verbunden sind. Alle fertigungstechnisch notwendigen Lager und die Transportwege zwischen den verschiedenen Firmen, die die einzelnen Fertigungsschritte ausführen, müssen entweder physikalisch oder kryptografisch abgesichert sein, um Manipulation oder Diebstahl von Halbfertigprodukten schon im Ansatz so schwierig wie möglich zu machen.

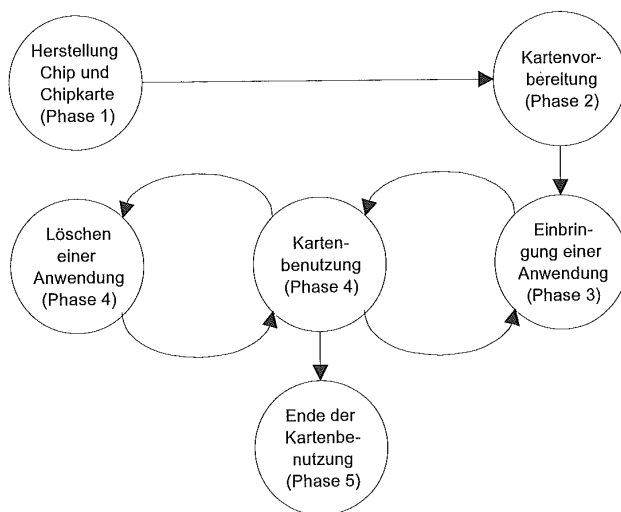


Bild 10.2 Der Lebenszyklus einer Chipkarte im Phasenmodell nach ISO 10 202-1. Das Laden und gegebenenfalls Löschen von Anwendungen, wie es bei Multiapplication-Chipkarten möglich ist, läuft in den Phasen 3 und 4 ab.

Begleitend zu allen Fertigungsschritten, muss natürlich eine entsprechende Qualitätssicherung stattfinden. Da es sich bei Chipkarten üblicherweise um Produkte für sicherheitsrelevante Bereiche handelt, ist es mittlerweile üblich, die Nachvollziehbarkeit der Produktion im Sinne der Normenreihe ISO 9000 ff. zu gewährleisten. Dies bedeutet, dass zumindest alle Fertigungsschritte mit Chargen- und Chipnummern protokolliert werden müssen. Zu einem beliebigen Zeitpunkt nach der Produktion muss es möglich sein, die einzelnen Fertigungsschritte individuell für jede Chipkarte nachzuvollziehen. Dies ermöglicht bei auftretenden Fertigungsfehlern eine einfachere Analyse der Ursachen. Da alle Mikrocontroller ab der Halbleiterfertigung durch eine einzigartige Nummer Unikate sind, lässt sich diese Nachvollziehbarkeit auf der Basis der Chipnummern relativ einfach realisieren. Die fertigungstechnische Verfolgung wird entweder durch eine Fertigungsdatenbank oder durch Eintrag aller fertigungsrelevanten Daten im Chip realisiert. Die ISO 10 202-1 empfiehlt die Speicherung der Fertigungsdaten im Chip, was einen Vorteil gegenüber der Speicherung in einer Datenbank darstellt. Bei der Speicherung im Chip kann man dann die relevanten Fertigungsdaten ohne Zugriff auf eine Datenbank abfragen; allerdings ist dazu kostbarer Speicher im EEPROM des Mikrocontrollers notwendig.

Tabelle 10.2 Typische in den Chip eingetragene Fertigungsdaten. Diese Daten können nur einmal geschrieben und anschließend nur noch gelesen werden (WORM-Attribut der Datei bzw. des Datenobjekts)

Phase des Kartenlebenszyklus	typische Fertigungsdaten
Phase 1 – Herstellung von Chip und Chipkarte	• ID des Chipherstellers • ID der Fertigungslinie • Eindeutige Nummer des Chips • Chiptyp • ID des Einbetters des Chips ins Modul • Datum und Uhrzeit der Einbettung des Chips ins Modul
Phase 2 – Kartenvorbereitung	• ID des Initialisierers • ID der Fertigungsmaschine • Datum und Uhrzeit der Initialisierung
Phase 3 – Einbringung einer Anwendung	• ID des Personalisierers • ID der Fertigungsmaschine • Datum und Uhrzeit der Personalisierung

10.2 Phase 1 des Lebenszyklus im Detail

Die erste Phase des Lebenszyklus nach ISO 10202-1 lässt sich nochmals unterteilen: Einerseits ist die Erstellung des Chipkarten-Betriebssystems und die halbleitertechnische Fertigung der Mikrocontroller abgedeckt und der zweite Ablaufpfad umfasst die gesamte Technik der Herstellung des Kartenkörpers.

10.2.1 Erstellung des Betriebssystems und Herstellung der Chips

Betriebssysteme und Software für die Chipkarten-Mikrocontroller sind so komplex, dass ihnen hier ein ganzes Kapitel gewidmet wird, welches alle Aspekte im Detail erläutert.² Nicht zu unterschätzen ist aber die Tatsache, dass gerade bei der Chipherstellung ein wesentlicher Teil der sicherheitstechnischen Grundlagen für den gesamten späteren Lebenszyklus gelegt wird. Ein noch so hochwertiges und mit viel Kryptografie abgesichertes Betriebssystem nützt wenig, wenn aufgrund eines Fehlers bei der Halbleiterherstellung alle geheimen Daten aus dem Chip ausgelesen werden können.

Die Halbleiterproduktion ist im Übrigen ein geschützter Bereich, zu dem nicht jedermann Zutritt hat. Dies ist in den Reinräumen relativ einfach zu bewerkstelligen, da sie ohnehin nur über Schleusen betreten werden können. Doch auch hinsichtlich der Sicherheitstechnik ist dies wichtig, da nur auf diese Weise gewährleistet werden kann, dass bei der Chipherstellung bzw. nach dem Vereinzeln der Chips keine Halbleiterbauteile mit darin enthaltenen trojanischen Pferden in Software eingeschuggelt werden können. Was im Übrigen ein sehr ernst zu nehmender und relativ gefährlicher Angriff auf die Sicherheit von Chipkarten-Anwendung wäre.

² Siehe auch Kapitel 5 „Chipkarten-Betriebssysteme“.

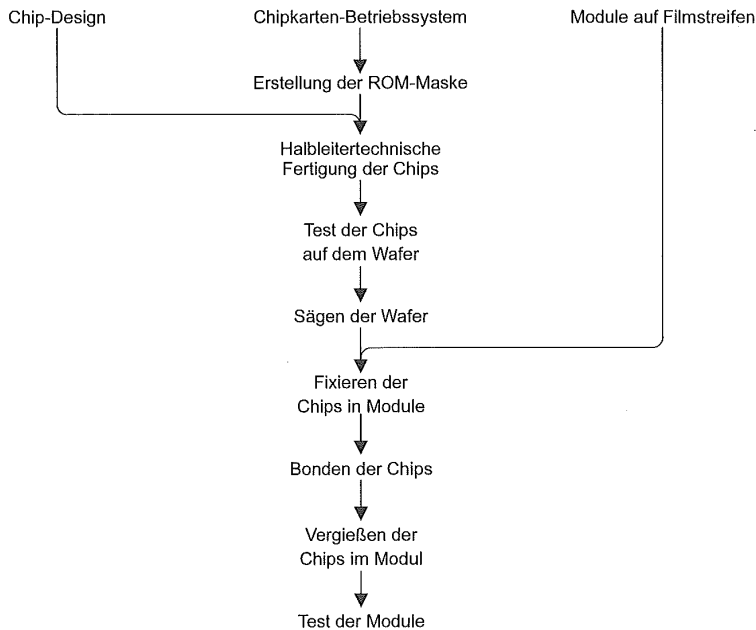


Bild 10.3 Chip- und Modulherstellung in Phase 1 im Lebenszyklus einer Chipkarte.

Chipdesign

Der geometrische Aufbau eines Chips für Speicher- oder Prozessorkarten sollte möglichst quadratisch sein, damit bei auftretenden Biegebeanspruchungen die Bruchgefahr des Chips minimiert ist. Ein vollkommener Biegeschutz des Chips durch ein extrem biegesteifes Modul ließe sich prinzipiell technisch realisieren, ist aber in der Praxis nicht gewollt. Solch ein steifes Modul würde bei der ständigen wechselseitigen Biegebeanspruchung der Karte über kurz oder lang zum Bruch der Karte führen.

Die für den Chip verwendeten halbleitertechnischen Komponenten wie CPU oder numerischer Koprozessor oder die unterschiedlichen Speicherarten sind in der Regel sicherheitstechnisch modifizierte Standardelemente.³ Oft verwendet man dazu Halbleiterkomponenten aus dem Automobilsektor, da diese für ähnlich harte Umwelt- und Zuverlässigkeitsbedingungen ausgelegt sein müssen. Diese müssen allerdings vollständig auf die Sicherheitsanforderungen von Chipkarten-Mikrocontrollern adaptiert und überarbeitet werden.

Beim Prozess des Chipdesigns wird nach der Funktionalspezifikation als erster Schritt eine generelle Chiparchitektur mit Blockschaltbild und Groblayout des zukünftigen Mikrocontrollers erstellt. Anschließend verfeinert man das globale Blockschaltbild schrittweise über Logikblöcke, Gatterfunktionen, Transistoren bis zu den geometrischen Strukturen für die einzelnen Belichtungsmasken. Dabei wird jeder Schritt durch Simulationen der Schaltungen und aufwendige Prüfprozesse begleitet. Dieser aus vielen Einzelschritten bestehende Prozess ist komplex, und es wird ein gehöriges Maß an Erfahrung für die optimale An-

³ Siehe auch Abschnitt 3.4 „Mikrocontroller für Chipkarten“.

ordnung der Schaltungselemente auf dem Chip benötigt. Schlussendlich werden auf einer Testfertigungslinie in einer Halbleiterfabrik Muster erstellt, die dann als erste Referenz genauestens vermessen und erprobt werden. Parallel dazu findet oft noch eine Sicherheitsbegutachtung statt, die allerdings frühestens mit den ersten richtigen Chips abgeschlossen ist.

Das Chipdesign ist ein Prozess, der Monate bis zu einem Jahr dauern kann, bis ein voll funktionsfähiger und alle Anforderungen erfüllender Chip für die Massenproduktion bereitsteht. Dieser Aufwand ist der Grund, warum die Generationenfolge bei Chipkarten-Mikrocontrollern zwischen zwei und drei Jahre beträgt. Der Tribut für größere Änderungen am Chip führt auch dazu, dass die wesentlichen Modifikationen zumeist Chip-Shrinks zur besseren Flächennutzung der Wafer oder kleinere Verbesserungen und Erweiterungen der Hardware sind.

Chipkarten-Betriebssystem

Die Softwareerstellung für ein Chipkarten-Betriebssystem und die darauf aufbauende Anwendung muss wegen der geringen Speicherkapazität der Mikrocontroller entweder in der Programmiersprache Assembler oder im hardwarenahen C durchgeführt werden. Diese sehr maschinennahen Programmiersprachen wirken sich natürlich ungemein verlängern auf die Dauer und somit auf die Kosten der gesamten Softwareentwicklung aus.

Die Tests der zum Großteil im ROM des Mikrocontrollers befindlichen Software sind sehr aufwendig und umfassend, da sich Fehler in dieser Software nach der Chipproduktion fast nicht mehr beseitigen lassen.⁴ Es wird dabei immer eine so genannte ROM-Maske entwickelt, d. h. eine Software, die sich im späteren ROM der Mikrocontroller befindet und in der eine Änderung nicht mehr möglich ist. Softwarefehler, die erst in einem der folgenden Fertigungsschritte erkannt werden, verursachen eine Wiederholung aller vorangegangenen Schritte.

Um den vorhandenen Speicherplatz des Mikrocontrollers möglichst gut zu nutzen, muss der Programmcode an den jeweiligen Chiptyp individuell angepasst werden. Die Programme sind deshalb nur mit zusätzlichem Aufwand auf andere Chiptypen portierbar. Die Dauer der Entwicklung für eine komplette ROM-Maske bewegt sich daher in einem Bereich um die 9 Monate. Sie kann jedoch auch deutlich geringer ausfallen, falls auf vorhandenen Programmcode (Software-Bibliotheken) zurückgegriffen werden kann. Ist die ROM-Maske fertig entwickelt, findet die offizielle Übergabe an einen Halbleiterhersteller statt.

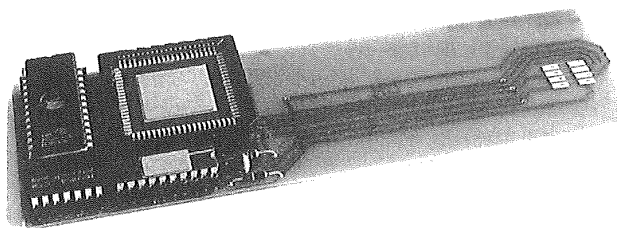


Bild 10.4 Miniemulator einer Chipkarte, bei dem das maskenprogrammierte ROM durch ein austauschbares EPROM im DIL-Gehäuse ersetzt ist. Der große Chip ist ein Mikrocontroller für Chipkarten, bei dem alle Busse frei zugänglich sind (Bond-out-Chip).

⁴ Siehe auch Kapitel 9 „Qualitätssicherung und Test“.

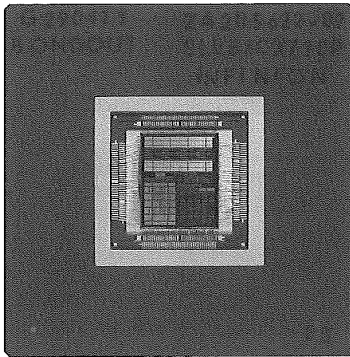


Bild 10.5 Beispiel für einen Bond-out-Chip zur Verwendung in Emulatoren für Chipkarten-Mikrocontroller. (Infineon)

ROM-Maske und halbleitertechnische Fertigung der Chips

Der Halbleiterhersteller erstellt aus der erhaltenen Software, die ihm auf einem EEPROM, einer Diskette oder per DFÜ übergeben wurde, eine Belichtungsmaske für das ROM des Mikrocontrollers. Die Maske, die den Programmcode enthält, wird von den Betriebssystemdesignern ROM-Maske oder oft nur Maske genannt. Werden die Strukturen bei der Abbildung auf den Wafer verkleinert, dann wird die ROM-Maske als Retikel bezeichnet. Die ROM-Maske ist aber nur eine Fotomaske von ca. 20, die zur Herstellung der Mikrocontroller notwendig sind. Der halbleitertechnische Aufbau der späteren Mikrocontroller findet auf entsprechend vorbereiteten hochreinen Siliziumscheiben, Wafer genannt, statt. Der Durchmesser der Wafer im Chipkartenbereich beträgt zurzeit meistens 6 bis 8 Zoll (= 15,2 – 20,4 cm). Auf einen 6-Zoll-Wafer passen bei 0,8 μm -Technologie ungefähr 700 Mikrocontroller.

Der Trend zukünftiger Fertigungsprozesse in der Halbleiterei geht jedoch in Richtung größerer Wafer und kleinerer Strukturbreiten. Man kann davon ausgehen, dass sich in den nächsten Jahren Wafer mit einem Durchmesser von 12 Zoll (= 30,6 cm) in 0,13 μm -Technologie als Standard in der Herstellung von Mikrocontrollern für Chipkarten etablieren. Die Kosten für eine Halbleiterfabrik dieser Technologiestufe liegen in der Größenordnung von etwa einer Milliarde Euro.

Noch vor einigen Jahren wurden Fotomasken für den gesamten Wafer verwendet, sodass die 700 Mikrocontroller alle auf einmal belichtet werden konnten. Dabei kam üblicherweise eine Kontaktbelichtung zum Einsatz. Mit kleiner werdenden Strukturbreiten auf dem Chip ist dies aber nicht mehr möglich, da die Ausbeute sonst nicht mehr akzeptabel wäre.

Bei allen neuen Produktionsverfahren beinhalten die Fotomasken nur mehr das Abbild eines einzigen späteren Chips und nicht mehr das des ganzen Wafers. Diese hoch empfindlichen Masken bestehen aus einer für ultraviolette Licht durchlässigen Quarzglas-Platte, auf der der auf dem Wafer abzubildende Chip mit Linien aus Chrom aufgetragen ist. Dazu ist auf der Quarzglas-Platte eine photoempfindliche Schicht aufgetragen, die mit einem Elektronenstrahlsschreiber belichtet wird. Anschließend wird sie entwickelt und durch Ätzen der unbelichteten Bereiche abgetragen.

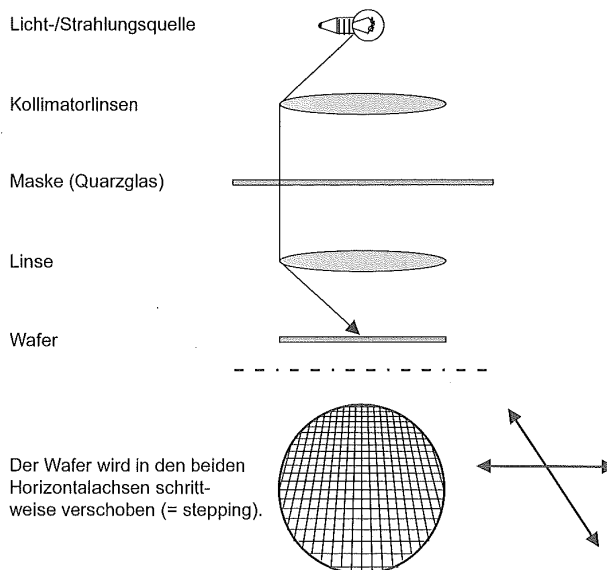


Bild 10.6 Prinzip der schrittweisen Belichtung einzelner Chips auf Halbleiterwafern mit einem Stepper.

Die Fotomaske wird im Maßstab 5 : 1 oder 10 : 1 vergrößert hergestellt, um bei der Belichtung des Wafers eine abbildungsverbessernde Verkleinerung zu ermöglichen. Die dabei zum Einsatz kommenden Belichtungsmaschinen (Jargon: „Stepper“) sind hochpräzise optische Geräte, die das Abbild der Maske auf dem Wafer auf Teile eines Mikrometers fokussieren und den Wafer in einer vergleichbaren Lagegenauigkeit verschieben können. Nach dem mit ultravioletttem Licht stattfindenden Belichtungsvorgang wird der Wafer um einen Positionsschritt verfahren, und es startet die Belichtung des nächsten Chips. So wird Schritt für Schritt (*stepping*) der ganze Wafer mit allen darauf befindlichen Mikrocontrollern belichtet.

Der gesamte Wafer ist mit einem lichtempfindlichem Lack (Fotoresist) versehen. An den belichteten Stellen wird dann der Lack durch Ätzen entfernt, und die darunter liegenden Waferflächen mit Fremdatomen dotiert. Nach mehreren Reinigungsschritten und erneutem Auftrag von fotoempfindlichem Lack ist der Wafer für die Belichtung mit der Nächsten von ca. 20 Masken bereit. Je nach Halbleiterhersteller und verwendetem Produktionsprozess be-

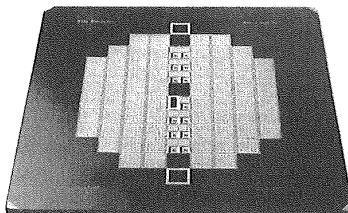


Bild 10.7 Fotomaske aus Quarzglas für die gleichzeitige Belichtung eines ganzen Wafers. (Philips)

nötigt man ungefähr 400 Prozessschritte und eine Durchlaufzeit von 6 bis 12 Wochen. Die reine Prozesszeit beträgt allerdings weniger als eine Woche. Diese in der Praxis extrem großen Durchlaufzeiten kommen vor allem durch die in der Massenproduktion bei Halbleitern übliche Warteschlangentechnik zustande.

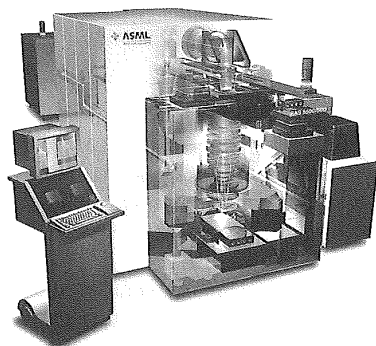


Bild 10.8 Schnittzeichnung eines Steppers für die schrittweise optische Belichtung einzelner Chips auf Halbleiterwafern. (ASM Lithography)

Aus Gründen eines ökonomischen Produktionsdurchlaufs müssen immer mehrere Wafer zu Chargen zusammengefasst durch die Halbleiterfertigung geschleust werden. Eine typische Charge besteht aus 12 Wafern, die oft übliche Mindestproduktionsmenge. Sie entspricht bei fast allen Halbleiterherstellern ungefähr 10 000 Chips. Kleinere Mengen als eine ganze Charge verursachen bei der Produktion einen erheblichen Mehraufwand, sodass die Fertigungskosten dann genauso hoch wären wie bei einer ganzen Charge. Manche Halbleiterfertigungen lassen jedoch so genannte „Shared-Batches“ oder „Multi-Project-Wafer“ zu, wobei verschiedenartige Mikrocontrollerchips mit unterschiedlichen ROM-Masken auf einem Wafer produziert werden. Dies ermöglicht dann auch kleinere Chargen als die obligaten 10 000 Stück. Allerdings lassen sich längst nicht alle Mikrocontroller in dieser Weise fertigen, und mitnichten ist jede Halbleiterfertigung fähig, „Shared-Batches“ einzuschleusen.

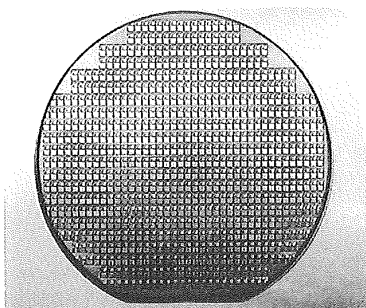


Bild 10.9 Ein 6 Zoll durchmessender Wafer mit ca. 700 Mikrocontrollern SLE 44C80. Die angefräste Fläche (*primary flat*) wird zur Kennzeichnung der Waferjustierung bei der Halbleiterproduktion verwendet. (Infineon)

Die Gesamtausbeute in der Halbleiterfertigung beträgt bei einem gut eingefahrenen Prozess etwa 80 %. So bleiben von den ursprünglichen 700 Mikrocontrollern auf einem Wafer für die weiteren Fertigungsschritte nur mehr ca. 560 Stück übrig. Bei neuen Herstellungsprozessen kann jedoch die Ausbeute durchaus über längere Zeit im 60 %-Bereich liegen, was sich somit sehr negativ auf den Durchsatz und auf die Rentabilität auswirkt.

Test der Chips auf dem Wafer

Im nächsten Fertigungsschritt werden die Mikrocontroller auf den Siliziumscheiben mit Metallspitzen kontaktiert und dann einzeln getestet. Dazu ist es notwendig, die 700 Mikrocontroller auf einem Wafer jeweils einzeln oder in Gruppen bis zu acht Stück zu kontaktieren und dann einen elektrischen Funktionstest durchzuführen. Da selbst in diesem Fertigungsschritt üblicherweise keine zusätzlichen Kontaktflächen für den Mikrocontroller existieren, können nur die später auch an der Chipkarte zur Verfügung stehenden fünf Kontakte benutzt werden.

Der Test der Funktionsbaugruppen auf dem Silizium ist allerdings wesentlich intensiver und umfangreicher als zu späteren Zeitpunkten, da sich die Mikrocontroller noch im so genannten Testmodus befinden. In diesem Modus kann man alle Speicher (RAM, ROM, EEPROM) ohne Einschränkung lesen und schreiben. Mikrocontroller, die aufgrund der Tests ausgesondert werden müssen, erhalten zur Identifizierung einen kleinen farbigen Punkt auf die Oberfläche. So können die nicht funktionsfähigen Bausteine in den folgenden Fertigungsschritten visuell erkannt und nach dem Trennen des Wafers einzelne Bauteile ausgesondert werden.

Weiterhin benutzt man den freien Schreibzugriff im Testmodus, um in diesem Fertigungsschritt chipindividuelle Daten in das EEPROM zu schreiben. Dies ist beispielsweise eine durchlaufende Nummer, die sich nicht wiederholen darf, also unique ist. Damit ist jeder Chip und somit jede zukünftige Chipkarte individualisiert. Neben einigen Sicherheitsaspekten hat dies den weiteren Vorteil, dass die Nachvollziehbarkeit nach ISO 9000 ff. aufgrund der einzigartigen Chipnummer stets gewährleistet ist.

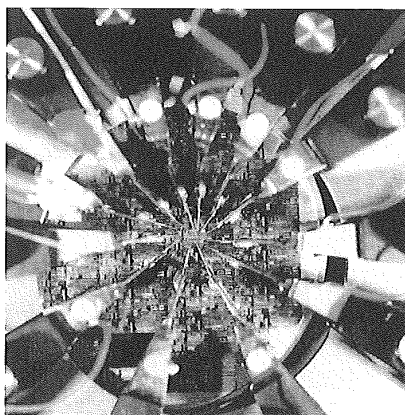


Bild 10.10 Test eines Mikrocontrollers auf dem Wafer. Dazu wird der Baustein mit Kontaktnadeln elektrisch kontaktiert und dann einzeln geprüft. (Philips)

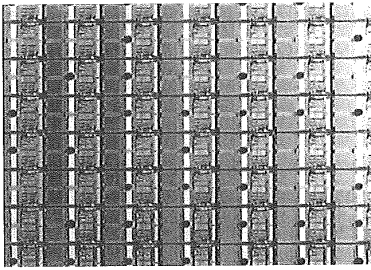


Bild 10.11 Ausschnitt eines Wafers mit halbleitertechnisch aufgebauten Mikrocontrollern. Die Punkte auf den einzelnen Chips identifizieren defekte Bausteine.

Sägen der Wafer

Nach dem Testen der Bausteine auf den Wafers müssen sie im nächsten Schritt vereinzelt werden. Dazu bringt man auf der Rückseite der Siliziumscheiben eine dünne Klebfolie auf, damit die einzelnen Chips beim Zerschneiden in ihrer Lage fixiert sind. Zum Einsatz kommen dabei Spezialsägen mit ca. 25 μm dicken Sägeblättern, die mit über 30000 Umdrehungen pro Minute rotieren. Die Siliziumscheiben zersägt man in einer Weise, dass sich auf den erhaltenen Teilen jeweils ein einzelner Mikrocontroller befindet. Die so gesägten maximal 25 mm² großen Siliziumkristalle mit den darauf aufgebauten Mikrocontrollern für die späteren Chipkarten werden Dice (Singular: Die) genannt. Nach dem Vereinzeln können die defekten und mit einem farbigen Punkt gekennzeichneten Dice von der Gutware abgetrennt und vernichtet werden.

Bis zu diesem Fertigungsschritt ist es nicht möglich, festzustellen, ob die Umsetzung der ROM-Software in das Silizium ohne Fehler vonstatten ging. Deshalb koppelt man in die

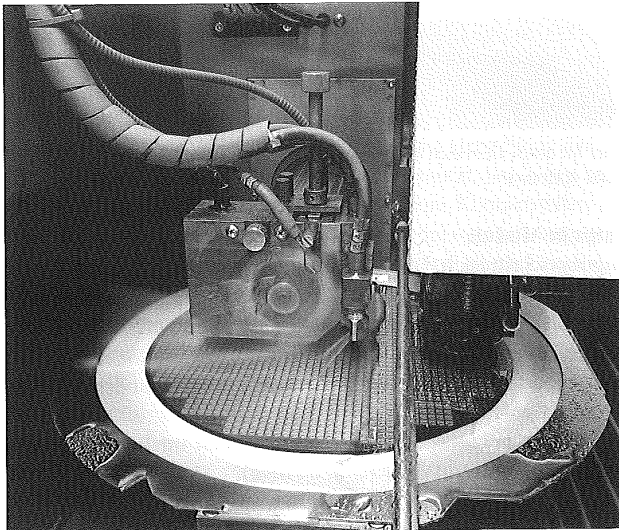


Bild 10.12 Zersägen eines Wafers in einzelne Dice. (Hitachi)

sem Fertigungsschritt etwa zehn Dice aus der Charge aus und baut sie in ein DIL-Gehäuse aus Keramik ein.

Der Softwarehersteller erhält diese ersten Muster und kann nun mit seinen Prüfmitteln feststellen, ob die Software im ROM des Mikrocontrollers fehlerfrei funktioniert. Auf diese Weise lässt sich der gesamte Chip testen. Sollte zu diesem Zeitpunkt ein Fehler in der Hard- oder Software festgestellt werden, muss man den weiteren Fertigungsprozess stoppen, und die ganze Charge hat nur mehr Materialwert. Sobald das Problem beseitigt wurde, ist ein Neustart des Produktionsvorgangs mit dem ersten Schritt beim Halbleiterhersteller „Erstellung der ROM-Maske“ notwendig. Der dadurch entstandene Zeitverlust ist auch durch beschleunigte Durchläufe in allen Fertigungsphasen nicht mehr aufzuholen.

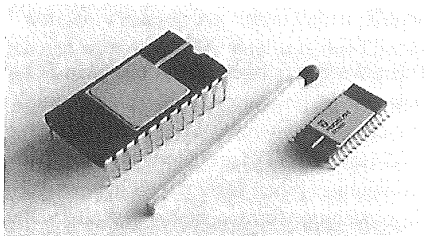


Bild 10.13 Beispiele für Mikrocontroller in unterschiedlichen Keramikgehäusen zum Test der Software.

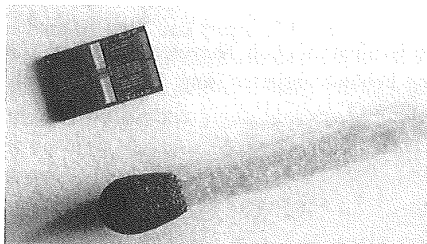


Bild 10.14 Beispiel für einen einzelnen Die im Vergleich zu einem Zündholz. Die Dicke des Die beträgt ungefähr 0,12 mm.

Fixieren der Chips in Module

Der nächste Fertigungsschritt nach dem Sägen der Wafer ist das Einbetten der Dice in Module. Diese erhöhen die Stabilität der sehr bruchempfindlichen Kristalle und sind mit den an der Oberseite angebrachten Kontaktflächen die spätere elektrische Verbindung zum Terminal.

Die übliche Anlieferungsform für Chipmodule sind Spulen mit 35 mm breiten Kunststoffstreifen mit Randperforation, auf denen paarweise nebeneinander zwei Module aufgebracht sind. Auf eine Spule passen abhängig von der Modulgröße zwischen 10000 und 20000 Module. Der 35 mm breite Kunststoffstreifen heißt im Insiderjargon schlicht und einfach „Tape“ und die Lieferform COT (*chip on tape*).

Die Breite des Streifens ist im Übrigen das Standardformat für Filme in der Kino- und Fototechnik. Der Grund für dieses Format liegt in den Anfängen der Chipkarten-Fertigungstechnik begründet. Um damals kostengünstige Transport- und Verpackungsmöglich-

keiten zu nutzen, die möglichst wenige Neuentwicklungen erforderten, verwendete man das 35-mm-Filmformat als Grundträger. Dadurch konnten viele der am Markt erhältlichen Spul- und Wickelgeräte für Filme verwendet werden. Da eine Umstellung auf andere Formate ab einem bestimmten Verbreitungsgrad nicht mehr durchführbar war, belässt man bis heute das 35 mm breite Trägermaterial.

Die Dice werden mit dem Siliziumträgermaterial nach oben, also zur Unterseite des Moduls hin, mit einem Kleber irreversibel verbunden. Damit lassen sich dann in weiteren Fertigungsschritten die Dice mit den korrespondierenden Modulkontaktflächen auf der Unterseite elektrisch verbinden.

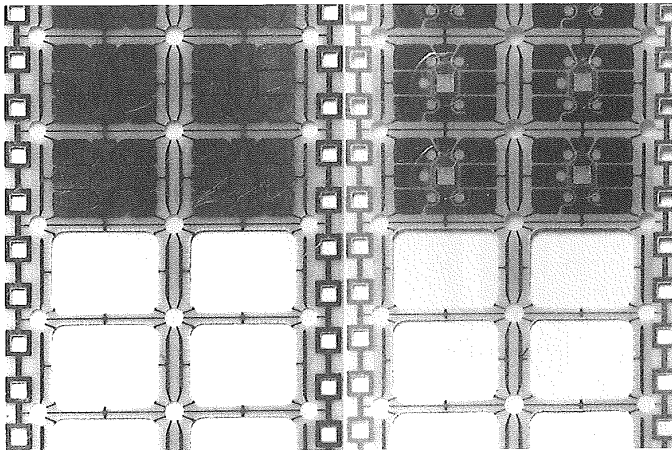


Bild 10.15 Beispiel für 35 mm breite Filmstreifen mit paarweise darauf angebrachten Modulen. Das linke Bild zeigt die Vorderseite und das rechte Bild die Rückseite des Filmstreifens. In den unteren Teilen der Bilder kann man die Lücken von bereits ausgestanzten Modulen erkennen.

Bonden der Chips

Nachdem die Dice in die Module eingeklebt sind, folgt die Herstellung der elektrischen Verbindung mit der Unterseite der Kontaktflächen. Dazu benutzt man einen sehr dünnen Golddraht, der sowohl an den dafür vorgesehenen Aluminium-Kontaktstellen auf den Dice als auch an den entsprechenden Kontaktpunkten auf der Modulunterseite angeschweißt wird.

Damit es bei starken Temperaturschwankungen nicht zu einem Bruch des Drahtes kommen kann, ist er in einem Bogen geführt. Dieser Bogen darf allerdings nicht zu groß sein, da die Bonddrähte sonst beim späteren Vergießen nicht mehr voll abgedeckt würden und damit eine erhöhte Korrosionsgefahr gegeben wäre.

Vergießen der Chips in den Modulen

Nach dem Bonden wird die Rückseite der Module und der montierten Dice mit einem schwarz eingefärbten Epoxidharz vergossen. Dieses Harz schützt den empfindlichen Kristall vor Umwelteinflüssen wie Feuchtigkeit, Torsion oder Biegung. Die Vergussmasse ist deshalb lichtundurchlässig, da Halbleiterbauelemente in der Regel sehr empfindlich gegenüber Licht und elektromagnetischen Wellen in den angrenzenden Spektralbereichen sind.

Anschließend wird der Trägerstreifen mit den Modulen auf große Filmspulen aufgewickelt und dann in Kartons verpackt. Für Kleinserien ist es auch noch möglich, die Module in Kunststoffmagazinen einzeln zu verpacken, doch wird dies für größere Stückzahlen vermieden, da beim Moduleinbatter dadurch die maschinelle Bearbeitbarkeit erschwert wird.

Sollen neue Mikrocontroller in den Markt eingeführt oder veränderte Chiphardware getestet werden, dann ist das Vergießen der Module oft schon der letzte Fertigungsschritt. Danach durchläuft man die entsprechenden Tests und Qualifizierungsstufen, und erst wenn dabei keine Fehler auftreten, startet man eine neue Charge mit entsprechend angepasster Software für die Massenproduktion. Ähnlich verhält es sich mit komplett neuen Betriebssystemen, bei denen man auch an diesem Punkt die Hardwareproduktion stoppt und dann über Wochen oder manchmal Monate die notwendigen Qualifizierungen vornimmt. Wenn nötig, muss man dann mit einem verbesserten Betriebssystem die Korrekturschleife noch einmal durchlaufen.

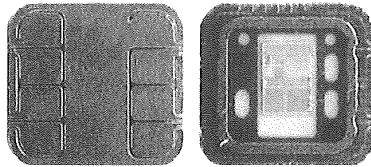


Bild 10.16 Beispiel für die Geometrie der Vorder- und Rückseite eines Moduls.

Test der Module

Die vorhergehenden Fertigungsschritte Sägen, Fixieren, Bonden und Vergießen haben zur Folge, dass dabei zwischen 3 % und 7 % der Dice unbrauchbar werden. Deshalb ist vor dem Verpacken und Ausliefern ein zusätzlicher Testschritt die Regel. Dazu müssen die Module an den Kontaktflächen an der Moduloberseite elektrisch mit einem Tester verbunden werden. Als Erstes schaltet dieser den Mikrocontroller durch Durchbrennen einer Polysiliziumsicherung vom Testmodus in den Usermodus oder schreibt einen speziellen Bytewert auf eine bestimmte Adresse im EEPROM. Damit ist es von nun ab nicht mehr möglich, von außen lesend oder schreibend auf den Speicher zuzugreifen, ohne die definierten Sicherheitsbedingungen zu erfüllen.

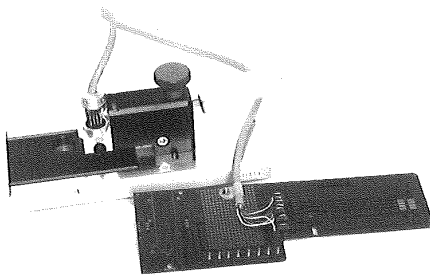


Bild 10.17 Adapter für Module auf Tape zu der üblichen ID-1 Größe.

Der Testrechner versucht dann, nach einer ISO-Anschaltsequenz einen gültigen ATR zu detektieren. Ist dies möglich, testet er mit den in der Maskensoftware integrierten Kommandos die Chiphardware. Verlaufen alle diese Tests positiv, haben die vorherigen Produktionsschritte bei dem getesteten Modul zu keinem Ausfall geführt, und es kann in eine Chipkarte eingebaut werden.

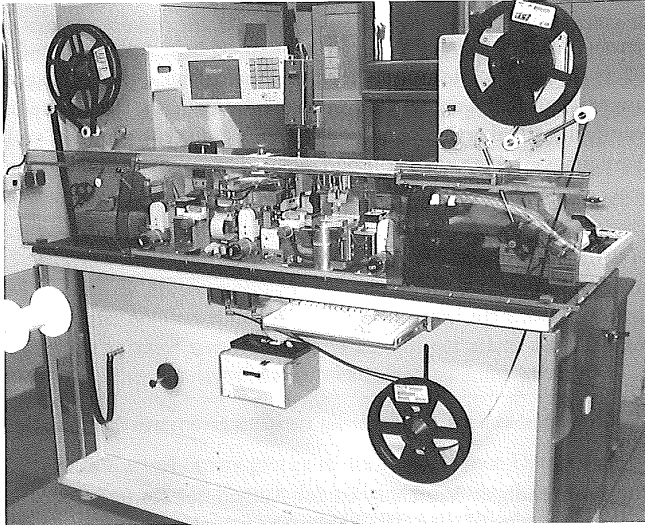


Bild 10.18 Maschine zur Wareneingangsprüfung von auf Tape montierten Modulen (d. h. COT – *chip on tape*).

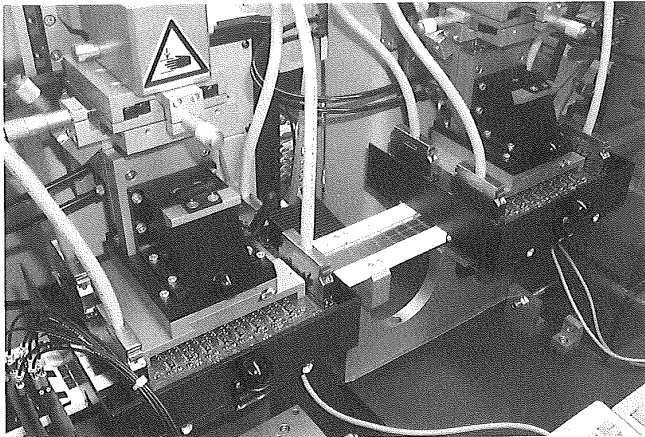


Bild 10.19 Detailfoto einer Maschine zur Wareneingangsprüfung von Chipmodulen. Dabei werden durch die beiden Kontaktköpfe jeweils 16 Module gleichzeitig kontaktiert und geprüft. Maschinen dieser Art können auch für die Initialisierung von Mikrocontrollern für Chipkarten verwendet werden.

10.2.2 Herstellung der Kartenkörper ohne integrierte Spule

Kartenkörper für Chipkarten ohne im Kartenkörper integrierte Spule können prinzipiell in drei unterschiedlichen Verfahren in der Massenproduktion hergestellt werden. Diese unterscheiden sich bezüglich Haltbarkeit, Oberfläche und möglichen Kartenelementen. Viele Kartenhersteller bieten oft nur eine Auswahl und nicht die ganze Palette der Herstellungsverfahren an.

Laien sehen oft die Herstellung von Kartenkörpern als unkomplizierte und leicht zu beherrschende Technologie. Schließlich müssen im Prinzip nur einige Kunststofffolien gestanzt und zusammengeklebt werden. Dies ist aber keinesfalls so. Die Herstellung von qualitativ hochwertigen Kartenkörpern in einer Massenproduktion basiert auf vielfältigen und komplexen Fertigungsschritten und erfordert eine hervorragende Beherrschung der notwendigen chemischen Prozesse bei Kunststoffen und dazugehörigen Farben.

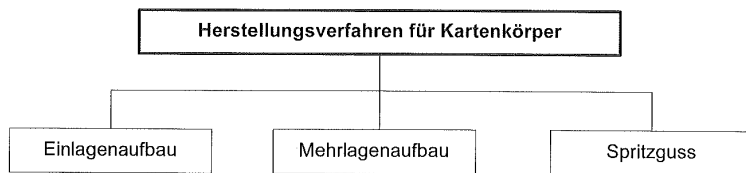


Bild 10.20 Klassifizierungsbaum der grundlegenden Herstellungsverfahren für Kartenkörper aus Kunststoff.

Das technisch aufwendigste Verfahren ist der Aufbau des Kartenkörpers aus miteinander heiß verklebten Kunststofffolien. Diesen Aufbau bezeichnet man als Mehrlagenaufbau und die Heißverklebung der Folien unter hohem Druck als Laminierung. Die Dicke der für das Karteninnere verwendeten so genannten Kernfolien (*core foils*) variiert zwischen 100 µm und 600 µm, die Dicke für die äußeren Deckfolien (*overlay foils*) zwischen 25 µm und 300 µm. Ein Kartenkörper in diesem Aufbau lässt große Gestaltungsvielfalt bei den Kartenelementen zu, ist sehr widerstandsfähig und bietet zusätzlich die Möglichkeit, zwischen den Folien Sicherheitsmerkmale einzufügen. Dies ist beispielsweise beim MM-Verfahren für die deutsche ec-Karte der Fall.⁵

Der Einlagenaufbau des Kartenkörpers aus einer einzigen 800 µm dicken Monofolie ist die vereinfachte Variante des Mehrlagenaufbaus. Dieses Verfahren ist kostengünstiger, die Karten sind jedoch etwas weniger widerstandsfähig als beim Mehrlagenaufbau und lassen vor allem viel weniger Gestaltungsmöglichkeiten bei den Kartenelementen zu. Beispielsweise können dann keine transparenten Deckfolien als Abrieb- und Kratzschutz über bedruckte Elemente auflaminiert werden.

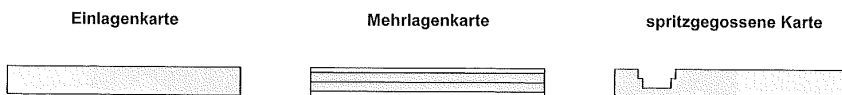


Bild 10.21 Überblick der üblichen Kartenaufbauten. Die Mehrlagenkarte setzt sich aus den äußeren Deckfolien (*overlay foil*) und den Kernfolien (*core foils*) zusammen.

⁵ Siehe auch Abschnitt 3.1.2 „Kartenelemente und Sicherheitsmerkmale“.

Die dritte Möglichkeit, einen Kartenkörper auf Kunststoff zu erzeugen, ist die Spritzgusstechnik. Sie führt dem Prinzip nach zu einem Kartenkörper in Einlagenaufbau mit all seinen Vor- und Nachteilen. Allerdings gibt es einige kleine, aber signifikante Abweichungen davon. Durch die Einlage einer ca. 80 µm dicken bedruckten Folie in die Spritzgussform können Spritzgusskarten bereits beim Spritzgießen mit einem Druck versehen werden. Dieses so genannte In-Mold-Labeling ist freilich im Vergleich zu Sieb- oder Offsetdruck mit einigen Einschränkungen in der Gestaltung und Verwendung von Druckfarben verbunden, doch hat es den Vorteil, dass die Kartenkörper nach dem Spritzguss nicht mehr durch eine Einzelkartendruckmaschine müssen. Eine weitere Eigenschaft der Spritzgusstechnik ist es, dass die Kavität für das Chipmodul während des Spritzgießens geformt wird und nicht noch anschließend ausgefräst werden muss. Es gibt auch bereits neuere Verfahren, bei denen das Chipmodul in die Spritzgussform eingelegt und so bereits beim Spritzgießen im Kartenkörper verankert wird. Mit dieser Technik erübrigen sich viele der im Folgenden beschriebenen Fertigungsschritte.

Übliche Spritzgussmaschinen haben eine Kapazität von ungefähr 2000 Kartenkörpern pro Stunde. Trotzdem ist die vermeintlich so kostengünstige Spritzgusstechnik bei großen Stückzahlen über einer Million Karten in der Regel teurer als Karten, die aus großen Bogen Kunststofffolien gestanzt werden. Bedingt ist dies vor allem durch die zeitaufwendige Handhabung der einzelnen Karten, die einer der Hauptkostenfaktoren ist.

Bedrucken der Folien

Beim Bedrucken des Kartenkörpers gibt es in der Regel keinen Unterschied zwischen Kartenkörper in Mehrlagen- oder Einlagenaufbau. Beim Bogendruck werden große Bögen aus Kunststofffolie bedruckt, aus denen man anschließend die Chipkarten stanzt. Eine übliche Größe des Bogens ist die Zusammenfassung von 21 bis 48 Drucken, d. h. Karten, zu einem Nutzen. Dieser Nutzen läuft dann in einem oder mehreren Arbeitsgängen durch die einzelnen Farbwerke einer Sieb- oder Offsetdruckmaschine. Für die Vorder- und Rückseite des Kartenkörpers ist jeweils ein separater Druck erforderlich.

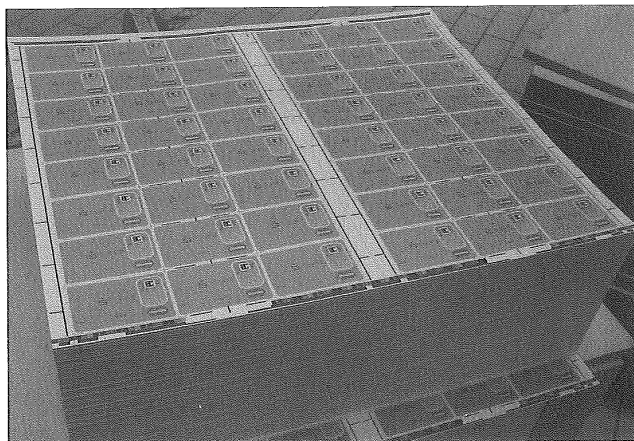


Bild 10.22 Aufeinandergestapelte bedruckte Nutzen mit je 48 Kartenkörpern.

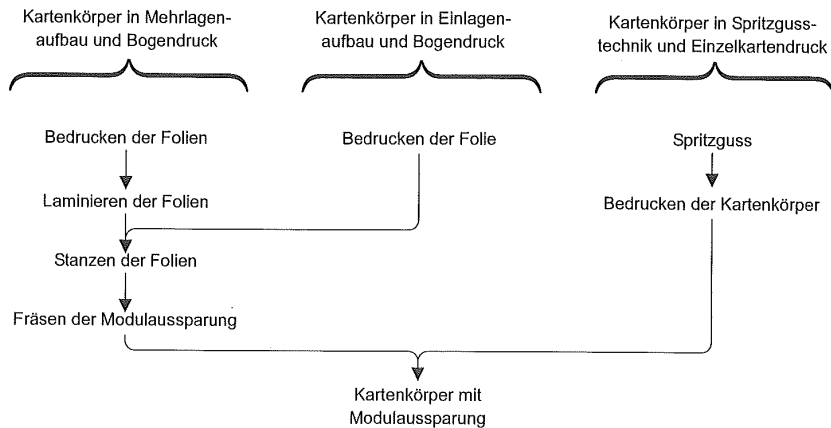


Bild 10.23 Herstellung der Kartenkörper für Chipkarten in der Phase 1 des Lebenszyklus einer Chipkarte.

Bei den für Karten eingesetzten Druckverfahren wird grundsätzlich zwischen Siebdruck und Offsetdruck unterschieden. Im Offsetdruck können feinere Details als im Siebdruck auf die Karte gedruckt werden. Die dabei verwendeten Farben härten unter UV-Licht aus. Da die UV-Bestrahlung sofort nach dem eigentlichen Druckvorgang stattfindet, hat man den Vorteil, dass die bedruckten Karten nach dem Druck sofort stapelbar sind. Allerdings können auf UV-härtende Lacke keine Magnetstreifen oder Hologramme im Hot-Stamp-Verfahren⁶ aufgebracht werden, da diese Lacke keine thermoplastischen Eigenschaften haben. Ähnlich verhält es sich mit Inlettfolien für laminierte Kartenkörper, da der Laminier-

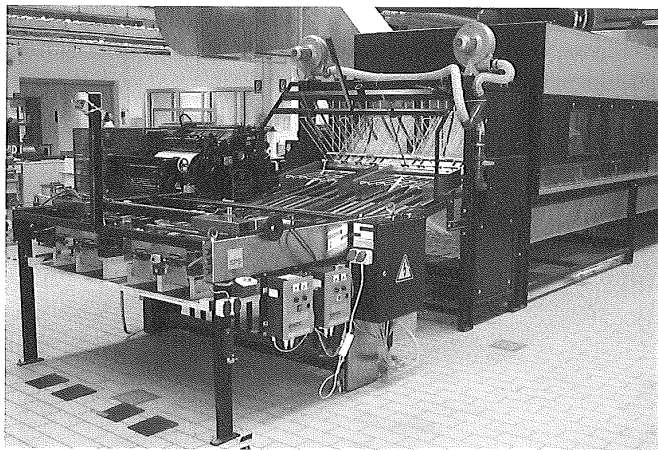


Bild 10.24 Siebdruckmaschine für den Bogendruck, bei dem beispielsweise ein Nutzen mit 48 Kartenkörpern gleichzeitig bedruckt werden kann.

⁶ Heißklebetechnik, die einen thermoplastischen Untergrund voraussetzt.

gang thermoplastische Eigenschaften der verwendeten Folien voraussetzt. Dafür verwendet man das Siebdruckverfahren, dessen Farben durch Verdunstung einer der Komponenten aushärten und sich thermoplastisch verhalten. Auf diese bedruckten Oberflächen lassen sich dann ohne weiteres zusätzliche Kartenelemente aufbringen. In der Praxis kombiniert man oft beide Druckverfahren. So druckt man die großen einfachen Flächen und den Untergrund für Magnetstreifen und Hologramm im Siebdruck. In einem zweiten Arbeitsschritt werden dann kleinere Details im Offsetdruck aufgebracht, deren Feinheit mit dem Siebdruck nicht zu erreichen wäre.

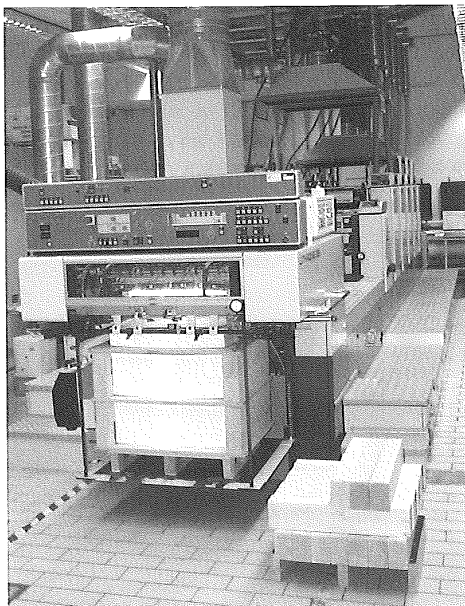


Bild 10.25 Maschine für Offsetdruck von Bögen.

Zusammenfassend kann man feststellen, dass der Offsetdruck ideal für farbige Motive in hoher Auflösung und großer Stückzahl geeignet ist. Müssen jedoch auf die Kartenoberfläche zusätzliche Funktionselemente wie etwa Hologramme irreversibel aufgebracht werden, dann muss entweder die ganze Karte oder zumindest der Untergrund des Hologramms mit einer thermoplastischen Farbe in Siebdruck bedruckt werden. Farben für Offsetdruck können dazu nicht verwendet werden, da sie unter UV-Licht härten und nicht thermoplastisch sind.

Als drittes Druckverfahren kommt manchmal noch der Thermotransferdruck zur Anwendung. Bei dieser Technik werden Teile von farbigen Folien durch Hitze abgelöst und auf weiße Karten aufgebracht. Die farbigen Folien verbinden sich fest mit der Oberfläche der Karte. Oft wird dieses Verfahren auch als reiner Schwarz-Weiß-Druck zum Aufbringen von Seriennummern benutzt. Prinzipiell lassen sich damit aber alle Farben und auch Farbverläufe darstellen. Das Verfahren ist allerdings langsam, teuer und findet vor allem

für Kleinserien in Desktop-Personalisierungsmaschinen Anwendung. Die Auflösung beträgt bis zu 300 dpi. Ein weiterer Nachteil ist, dass die aufgetragenen Farben nur an der Oberfläche der Karte haften und sich beispielsweise durch Kratzen wieder ablösen lassen.

Der Nachteil der nur auf der Kartenoberfläche haftenden Farbe kann durch die Technik des Thermosublimationsdrucks umgangen werden. Dabei dringt die am Druckkopf auf annähernd 200 °C erhitzte Farbe in die oberste Kunststoffschicht des Kartenkörpers ein. Die Eindringtiefe beträgt maximal 5 µm, was ausreicht, damit dieser Druck kratzfest ist. Die anderen Merkmale des Thermosublimationsdrucks sind im Wesentlichen identisch mit dem Thermotransferdruck, so auch die hohen Druckkosten, was das Verfahren ebenfalls nur für kleinere Serien geeignet macht. Beide Druckverfahren sind jedoch vor allem für den Druck auf Karten mit entsprechend vorbereiteter Oberfläche geeignet, was ihre Verwendung einschränkt. Allerdings stellen sie eine durchaus sinnvolle Ergänzung zum Just-In-Time-Druck für kleine und mittlere Serien dar.

Tabelle 10.3 Überblick der geläufigsten Verfahren für das Bedrucken von Karten

Eigenschaften	Offsetdruck	Siebdruck	Thermotransferdruck	Thermosublimationsdruck
Bogendruck für Massenproduktion möglich	ja	ja	nein	nein
Auflösung	sehr gut	mittel	gut	gut
Oberflächendeckung der Farbe	gut	sehr gut	befriedigend	befriedigend
bedruckte Oberfläche laminierfähig	nein	ja	nicht üblich	nicht üblich
Druck kratzfest	nein	nur mittels überlaminierter Overlayfolien	nein	ja
kartenindividueller Druck möglich	nein	nein	ja	ja
Preis	niedrig	niedrig	hoch	hoch

Laminieren der Folien

Sind die Folien gedruckt, werden sie im Falle eines Kartenkörpers im Mehrlagenaufbau bei 100 °C bis 150 °C zusammenlaminiert und zusätzlich die geforderten Merkmale integriert. Gegen Verkratzen und Abrieb schützt man die bedruckten Folien für Vorder- und Rückseite durch zusätzlich auflaminierte transparente Deckfolien, oft auch als Overlayfolien bezeichnet. Abhängig vom Kundenauftrag sind Unterschriftstreifen, Magnetstreifen und Sicherheitsmerkmale in den Mehrlagen-Kartenkörper während dieses Produktionsschrittes ebenfalls einzubetten oder aufzulaminieren.

Karten können durchaus mit unterschiedlich dicken Kunststofffolien für Vorder- und Rückseite gefertigt werden. Generell ist es jedoch aus mechanischer Sicht besser, einen symmetrischen Kartenaufbau zu wählen. Dies heißt, dass man für Vorder- und Rückseite gleich dicke Folien benutzt. Damit umgeht man die Problematik, dass es zu einem „Bimetalleffekt“ kommt und sich die Karte zu einer Seite hin verbiegt.

Stanzen der Folien

Nach dem Zusammenlaminieren der einzelnen Kunststoffbögen müssen sie auf das gewünschte Kartenformat gebracht werden. Dies wird durch einen Stanzvorgang erreicht. Die dazu benutzten Maschinen haben einen stündlichen Durchsatz von 4000 bis 8000 Karten. Der Grat, der an den Rändern von manchen Chipkarten zu sehen oder zu fühlen ist, rührt im Übrigen von abgenutzten Stanzwerkzeugen her.

Fräsen der Modulaussparung

Im Anschluss an die Herstellung des Kartenkörpers muss eine Aussparung für das Modul hineingefräst werden. Diese Aussparung wird „Kavität“ genannt. Es gibt auch Verfahren, bei denen die Folien vorgestanzt sind, sodass sich eine Aussparung ergibt, die beim Zusammenkleben genutzt wird, um darin das Modul einzubetten. Bei der Kartenkörperherstellung in Spritzgusstechnik ist die Modulaussparung ebenfalls schon durch die Spritzgussform vorgegeben. Doch bei Kartenkörpern aus einzelnen Folien wird in der Regel der Platz für das Modul aus dem Kartenkörper gefräst.

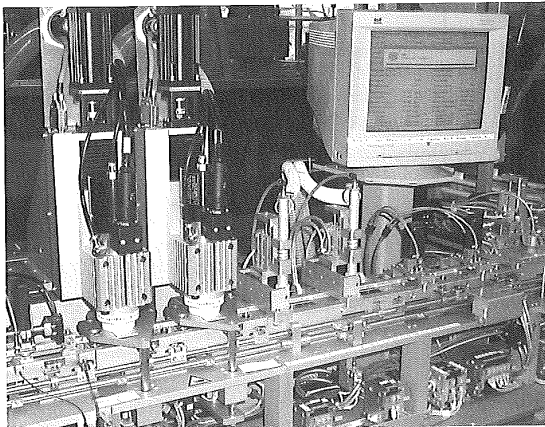


Bild 10.26 Fräsmaschine für die Kavität in Kartenkörpern. Die beiden links von der Bildmitte befindlichen nach oben ragenden Elemente sind elektrisch angetriebene Frässpindeln mit integrierter Frässpanabsaugung.

Da das Modul auf der Unterseite eine Erhebung an der Stelle des vergossenen Die hat, muss eine dazu passende Aussparung beim Fräsen des Kartenkörpers geformt werden. Eine einstufige Kavität ist bei den heute üblichen Modulformen meist ungünstig, weshalb die Kavität üblicherweise zweistufig oder sogar dreistufig gefräst wird. Die daraus resultierende größere Kontaktfläche zwischen Modul und Kartenkörper ermöglicht eine dauerhafte Verklebung. Zudem ist es mechanisch wesentlich günstiger, wenn das Modul nur an seinem Rand mit dem Kartenkörper fest verbunden ist und das an der Unterseite des Moduls befindliche Die keinen mechanischen Kontakt mit dem Plastik der Karte hat. Das Modul wird sozusagen „schwimmend“ eingebaut.

Als Erstes fräst man nun eine Aussparung in der Größe der Kontaktflächen und einer Tiefe, die der Höhe der Kontaktflächen entspricht. Danach wird in der Mitte der soeben er-

zeugten Aussparung eine zusätzliche Vertiefung für das vergossene Die eingefräst. Das Ergebnis ist eine Aussparung mit zwei Stufen.⁷

Die Fräsvorgänge müssen sehr exakt verlaufen, da an der tiefsten Stelle die Materialdicke der Karte nur noch ca. 0,15 mm misst. Vibrationen oder Schwingungen beim Fräsen können dazu führen, dass die Karte durchgefräst wird und dann nicht mehr zu gebrauchen ist. Wäre aber die Kavität nicht tief genug, würde anschließend das Modul an der Kartenoberseite überstehen, was jedoch nur innerhalb sehr enger Grenzen erlaubt ist. Dieser diffizile Fertigungsschritt findet in einer vollautomatischen Maschine mit Zufuhr und Abtransport der Kartenkörper in Magazinen statt. Der Durchsatz bewegt sich im Bereich von etwas über 1 000 Karten pro Stunde.

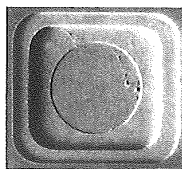


Bild 10.27 Beispiel für eine gefräste Modulaussparung im Kartenkörper, im Fachjargon auch „Kavität“ genannt.

Bedrucken der Kartenkörper

Das zweite drucktechnische Verfahren neben dem Bogendruck ist der Einzeldruck, bei dem bereits vereinzelte Karten bedruckt werden. Werden die Karten im Einzeldruck bedruckt, so geschieht dies immer vor dem Fräsen der Modulaussparung. Der Durchsatz von Druckmaschinen für Einzeldruck bewegt sich im Bereich bis zu 12 000 Karten pro Stunde. Eine weitere Variante des Einzelkartendrucks ist der Thermotransfer- und Thermosublimationsdruck in Desktop-Personalisierungsanlagen, bei denen der Durchsatz wesentlich geringer ist und bei maximal 300 Karten pro Stunde liegt.



Bild 10.28 Hintereinander angeordnete Serie von fünf Einzelkartendruckmaschinen für den Siebdruck auf einzelne Karten.

⁷ Siehe auch Abschnitt 3.2.2 „Chipmodule“.

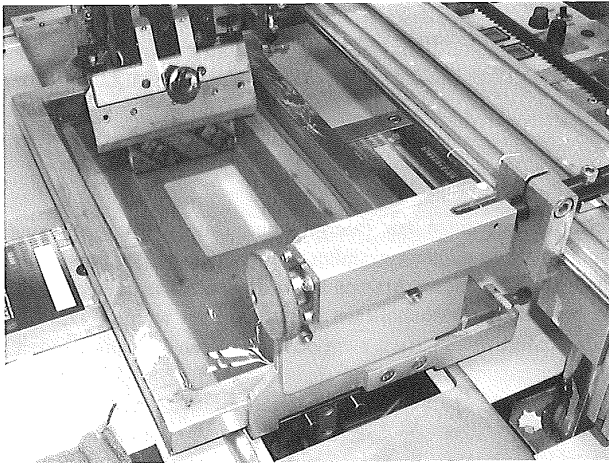


Bild 10.29 Detail einer Einzelkartendruckmaschine für Siebdruck.

Aufbringung von Kartenelementen auf den Kartenkörper

Nachdem die Kartenkörper auf Maß gebracht wurden, werden noch unterschiedliche Kartenelemente wie Hologramme oder Magnetstreifen aufgebracht. Die als Rollenware zugelieferten Hologramme werden dabei in einer Heißklebetechnik (Hot-Stamp-Verfahren oder Roll-

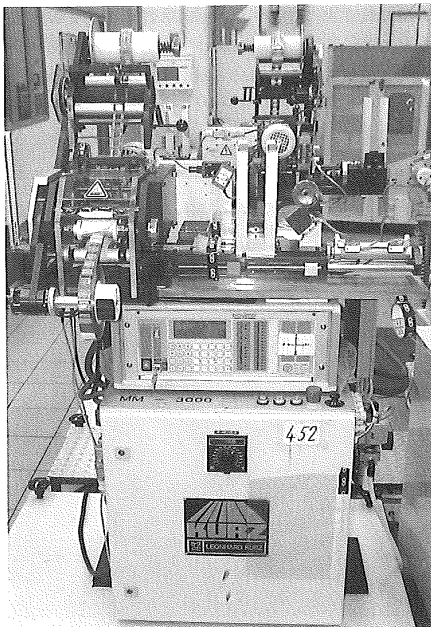


Bild 10.30 Maschine zur Aufbringung von Hologrammen auf Kartenkörpern mittels Heißprägeverfahren.

On-Verfahren) irreversibel mit dem Kartenkörper verbunden.⁸ Ein Entfernen ist dabei nur unter Zerstörung des jeweiligen Sicherheitsmerkmals möglich. Der Magnetstreifen wird entweder im Laminier-Verfahren oder im Hot-Stamp-Verfahren auf den Kartenkörper aufgebracht.

10.2.3 Herstellung von Kartenkörpern mit integrierter Spule

Kontaktlose Chipkarten benötigen zur Energie- und Datenübertragung eine Spule. Bei hohen Frequenzen kann diese so klein gestaltet werden, dass sie sich ins Chipmodul integrieren lässt. Deshalb kann bei dieser Art von kontaktlosen Karten ein vergleichbarer Fertigungsprozess wie bei den kontaktbehafteten Karten benutzt werden. Das Chipmodul mit der Spule wird dabei lediglich zwischen zwei oder mehreren Kunststofffolien einlaminiert oder in eine Kavität eingesetzt.

Allerdings benutzt man bei den meisten heutigen kontaktlosen Chipkarten niedrigere Frequenzen und benötigt deshalb Spulen mit größerem Durchmesser. Diese sind meist rechteckig mit gerundeten Ecken und ungefähren Abmessungen von 75 mm auf 45 mm. Solche Spulen sind damit nur etwas kleiner als der gesamte Kartenkörper im ID-1 Format. Üblicherweise besitzen sie vier Windungen, eine Induktivität von etwa 4 μH und einen elektrischen Widerstand von einigen wenigen Ohm. Die Ausnahme sind gedruckte Spulen, die einen Widerstand im Bereich von 300 Ohm aufweisen.

Für die Herstellung dieser speziellen Kartenkörper mit integrierter Spule muss der übliche Fertigungsprozess für Kartenkörper entsprechend den geänderten Anforderungen angepasst werden. Einige generelle Prinzipien ändern sich jedoch nicht. So werden auch kontaktlose Chipkarten meist zu Nutzen von 48 Stück auf großen Kunststoffbögen zusammengefasst und nicht einzelstückweise gefertigt, da dies deutlich kostengünstiger ist.



Bild 10.31 Querschnitt einer kontaktlosen Chipkarte mit in den Kartenkörper eingebettetem Modul und gewickelter Spule. Die sichtbare Kantenlänge des Chips im Innern des Moduls beträgt 2 mm. (Giesecke & Devrient)

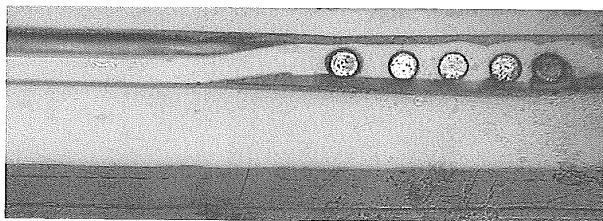


Bild 10.32 Detail vom Querschnitt einer kontaktlosen Chipkarte mit in den Kartenkörper eingebettetem Modul und gewickelter Spule. (Giesecke & Devrient)

⁸ Siehe auch Abschnitt 3.1.2 „Kartenelemente und Sicherheitsmerkmale“.

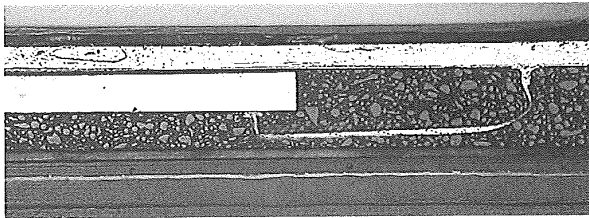


Bild 10.33 Detail vom Querschnitt eines in den Kartenkörper eingebetteten Chipmoduls für kontaktlose Chipkarten. Deutlich zu sehen ist der Chip und Spule elektrisch verbindende Bonddraht. (Giesecke & Devrient)

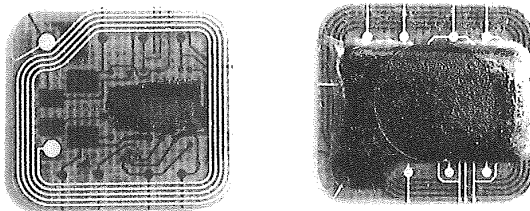


Bild 10.34 Vorder- und Rückansicht eines Moduls mit integrierter Spule für einen Mikrocontroller-Chip.

Kontaktierung von Spule und Chip

In der zurzeit üblichen Standard-Technologie wird der mit der Spule elektrisch zu verbindende Die auf einem Lead-Frame befestigt und galvanisch mit den beiden darauf befindlichen Kontaktflächen verbunden, die man dann zur Herstellung des elektrischen Kontaktes mit den beiden Enden der Spule benutzt. Der Grund für die Benutzung eines Lead-Frames ist die Tatsache, dass bei der elektrischen Verbindung mit der Spule nicht so genau positioniert werden muss, als wenn diese direkt mit dem Die kontaktiert würde. Allerdings ist dieses Verfahren teurer als die direkte Verbindung von Die und Spule, was eine sehr exakte Positionierung und hochwertige Verbindungstechnologie voraussetzt.

Bei in den Kartenkörper integrierten Spulen gibt es zwei Varianten, wie der Chip mit der Spule bzw. dem Lead-Frame elektrisch kontaktiert wird. Mit dem technisch geläufigen Verfahren des Wire-Bondings wird der Chip durch dünne Bond-Drähte verbunden. Wesentlich eleganter und auch kostengünstiger ist jedoch das Die-Bonding, bei dem der Chip so auf



Bild 10.35 Detail der elektrischen Verbindung eines kleinen Chips mit einer geätzten Spule durch Die-Bonding.

den Lead-Frame oder die Spule gepresst und dann verklebt wird, dass eine direkte elektrische Verbindung geschaffen wird. Dabei muss der Chip jedoch im Vergleich zum Wire-Bonding umgedreht werden, weshalb diese Technik oder manchmal auch der Chip als Flip-Chip bezeichnet wird. Bei beiden Verfahren wird nach der Herstellung der elektrischen Verbindung der Chip mit einem Schutzüberzug aus Kunststoff versehen.

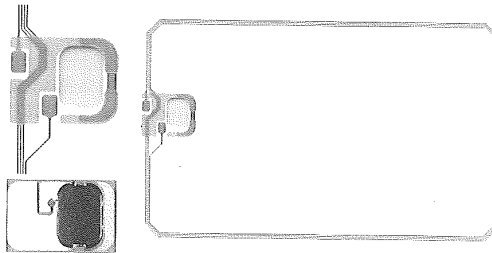


Bild 10.36 Rechts ist die Inlay-Folie einer kontaktlosen Chipkarte mit geätzter Spule zu sehen. Links oben befindet sich eine Ausschnittsvergrößerung der Modulaussparung sowie der Kontaktierungsflächen der Spule zum Modul. Darunter ist das dazugehörige Modul mit dem eingebetteten Chip abgebildet.

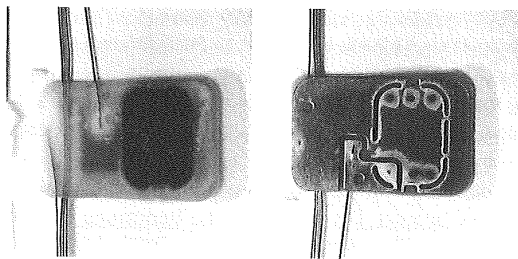


Bild 10.37 Vorder- und Rückansicht eines Lead-Frame-Moduls für kontaktlose Chipkarten und mit kontaktierter gewickelter Spule.

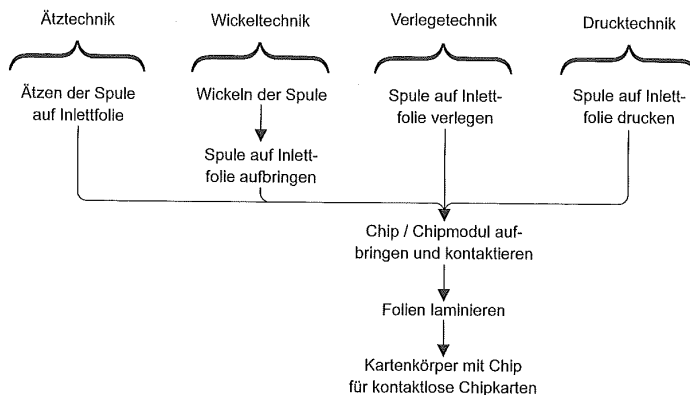


Bild 10.38 Herstellung von kontaktlosen Chipkarten in der Phase 1 des Lebenszyklus einer Chipkarte.

Geätzte Spule

Es gibt mehrere Möglichkeiten, eine Spule in einen Kartenkörper zu integrieren. Bei dem zuerst entwickelten Verfahren für die Herstellung von kontaktlosen Chipkarten werden mit $35\text{ }\mu\text{m}$ dickem Kupfer beschichtete und somit elektrisch leitende Kunststofffolien so geätzt, dass auf ihnen eine Spule entsteht. Der geätzte Verlauf der Leiterbahn hat ungefähre Abmessungen von $100\text{ }\mu\text{m}$ in der Breite. Nach dem Ätzen wird an den Anschlussstellen der Spule ein Chip platziert und elektrisch verbunden. Dann benutzt man dies im Sinne einer Inlettfolie und laminiert hinten und vorne eine Deckfolie auf. Die Chipkarte ist nun fertig. Diese Art von Karte wird als kontaktlose Chipkarte mit geätzter Spule (*etched coil*) bezeichnet.

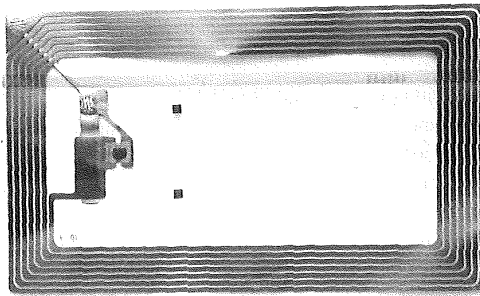


Bild 10.39 Spule für kontaktlose Chipkarten in Ätztechnik.

Gewickelte Spule

Eine Weiterentwicklung der geätzten Spule ist die gewickelte Spule (*wound coil*). Dazu wird ein $150\text{ }\mu\text{m}$ durchmessender Kupferdraht auf eine keilförmige Spulenform gewickelt, von der Form dann auf eine Inlettfolie geschoben und mit Hitze und Druck thermoplastisch mit der Inlettfolie verschweißt. Anschließend werden noch der Chip bzw. Chipträger aufgesetzt und die Folien laminiert. Diese Technik ist deutlich kostengünstiger und besser für große Stückzahlen geeignet als die der geätzten Spule.

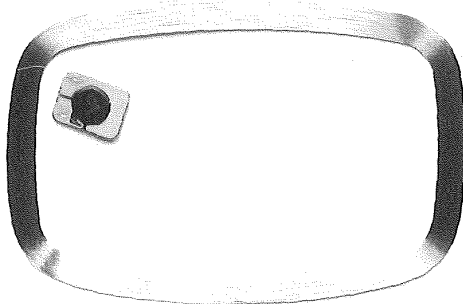


Bild 10.40 Gewickelte Kupferdrahtspule für kontaktlose Chipkarten mit einem Modul. Diese Technologie ist mittlerweile veraltet, zeigt jedoch sehr gut den Aufbau von Spule und Modul.

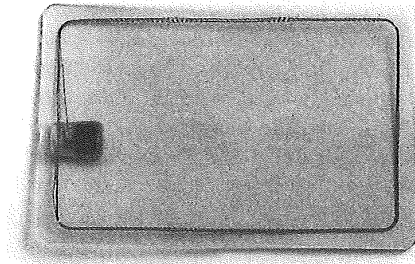


Bild 10.41 Beispiel für eine gewickelte Spule mit einem Lead-Frame-Modul in einem für Demonstrationszwecke transparent gehaltenen Kartenkörper.

Verlegte Spule

Ein anderes Verfahren, um eine Drahtspule herzustellen, ist die so genannte verlegte Spule (*embedded coil*). Die Funktionsweise ist relativ einfach: Ein Kupferdraht mit etwa $150\ \mu\text{m}$ Stärke wird direkt auf einer Kunststoff-Folie verlegt und gleichzeitig durch Ultraschall mit dem Kunststoff fest verbunden. Dazu verwendet man eine Sonotrode, eine mechanische Führung des Drahtes, die den Draht simultan beim Verlegen mit der Kunststoff-Folie ultraschallverschweißt. Anschließend verbindet man noch den Chip oder den Chipträger mit der Spule und laminiert eine Deckfolie auf.

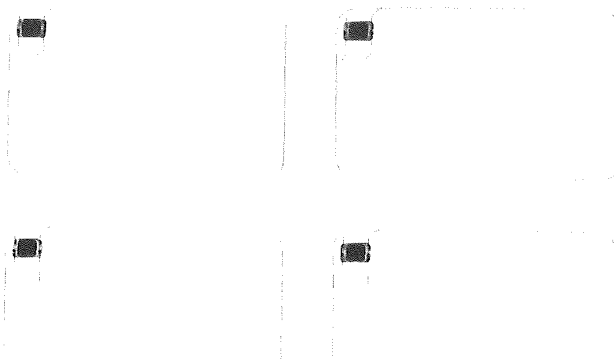


Bild 10.42 Spulen für kontaktlose Chipkarten auf einem Nutzen in Draht-Verlegetechnik.

Gedruckte Spule

Das technisch am weitesten entwickelte und für die Massenproduktion kostengünstigste Verfahren ist jedoch die gedruckte Spule (*printed coil*). Im Siebdruck und mit einer elektrisch leitfähigen Farbe werden auf eine Inlettfolie die Windungen einer Spule gedruckt. Siebdruck ist zu verwenden, um genügend Farbe aufzutragen, damit man eine Farbdicke von etwa $50\ \mu\text{m}$ erreicht. Der elektrische Widerstand der Spule darf nicht zu hoch werden. Nach dem Druck kontaktiert man den Chip durch Die-Bonding, vergießt ihn mit Epoxid-

harz und laminiert als letzten Fertigungsschritt eine Deckfolie als Schutz auf. Dieses Verfahren hat den großen Vorteil, dass es aufgrund des technisch einfachen Druckprozesses einen hohen Durchsatz hat und damit außerordentlich gut für die Produktion von großen Stückzahlen geeignet ist. Allerdings benötigt man ein erhebliches Know-how, um sowohl den Druck als auch das Die-Bonding in der benötigten Qualität zu erreichen.

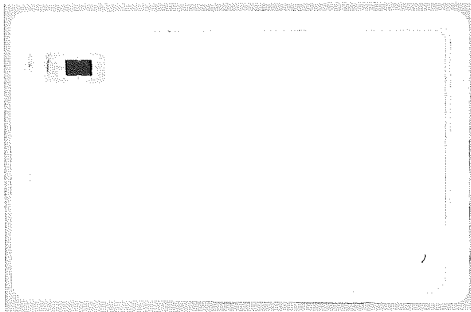


Bild 10.43 Spule für kontaktlose Chipkarten in Drucktechnik.

10.2.4 Zusammenführen von Kartenkörper und Chip

Die Implantierung der vom Halbleiter- oder Modulhersteller stammenden Module mit den beim Kartenhersteller vorfabrizierten Kartenkörpern ist der letzte fertigungstechnische Schritt, bei dem besonders die mechanischen Aspekte im Vordergrund stehen. Die dauerhafte Verankerung des Moduls in der Kavität des Kartenkörpers erfordert jedoch durchaus einiges an Fachwissen und ist nicht nur eine einfältig einfache „Einklebearbeit“.

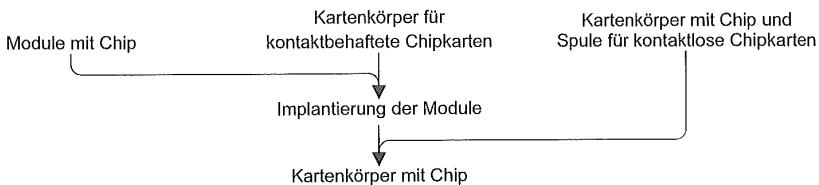


Bild 10.44 Implantierung der Module in Kartenkörper für kontaktbehaftete Chipkarten in der Phase 1.

Implantierung der Module

Unabhängig von den verschiedenen Herstellungsverfahren für Kartenkörper und der Erzeugung einer Aussparung für das Modul muss in einem nachfolgenden Fertigungsschritt das Modul in den Kartenkörper eingebettet werden.

Üblicherweise verwendet man doppelseitiges Klebeband mit einer Heißklebekomponente, um die Module im Kartenkörper zu befestigen. Die Klebung besteht nur an der Randauflagefläche des Moduls mit dem Kartenkörper. Das in der Modulmitte überstehende und vergossene Die bleibt frei, und das Chipmodul ist damit schwimmend befestigt. Dazu muss das Klebeband vorab so ausgestanzt und auf den 35 mm breiten Streifen mit den Modulen aufgebracht werden, dass nur der Rand der Module abgedeckt ist. Anschlie-

ßend vereinzelt man die Module und klebt sie mit dem aufgetragenen Klebeband in den Kartenkörper ein. Die drei Größen Druck, Temperatur und Zeit sind dabei ausschlaggebend für die Beständigkeit der Klebung.

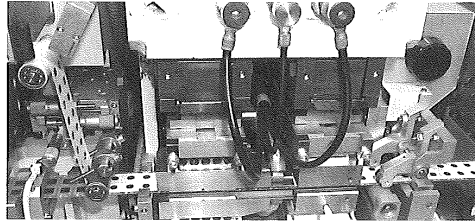


Bild 10.45 Maschine zum Auflaminieren eines Klebebandes auf die 35 mm breiten Streifen mit Modulen.

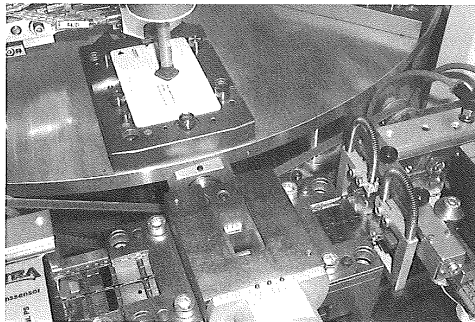


Bild 10.46 Eine nach dem Rundläuferprinzip arbeitende Maschine zur Implantation von Chipmodulen. Im Vordergrund sieht man die auf einen 35 mm-Streifen paarig angeordneten Module mit einer kleinen Stanze, von wo sie mit einem Sauger abgehoben und in die Kavität des Kartenkörpers gesetzt werden.

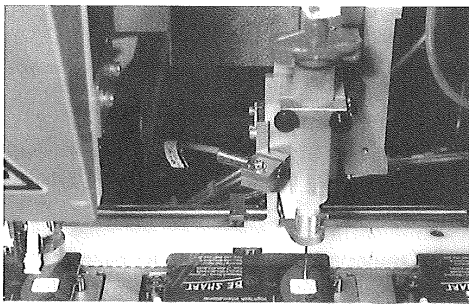


Bild 10.47 Einbringen von Flüssigkleber in die vorgefräste Modulaussparung (Kavität) am Implantier. (Mühlbauer)

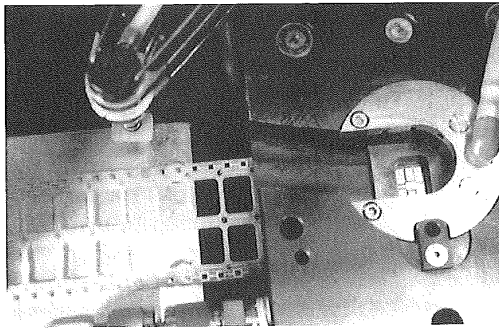


Bild 10.48 Ausstanzen des Moduls aus einem Tape am Implanter. (Mühlbauer)

Problematisch dabei ist, dass die Module eine kurzzeitige Temperaturerhöhung (≈ 1 Sekunde) auf ca. 180°C erfahren. Falls dieser Heißklebevorgang, der normalerweise ungefähr eine Sekunde benötigt und einiges an technischem Know-how erfordert, zu lange dauert, führt dies zur thermischen Zerstörung des Moduls. In jedem Fall bedingt die kurzzeitige Temperaturerhöhung eine künstliche Alterung des Chips, die aber im Normalfall keine nachteiligen Auswirkungen hat. Die in den Kartenfertigungen eingesetzten Implanter haben eine stündliche Kapazität von ungefähr 2000 Modulen, was einer Taktzeit von 1,8 Sekunden pro Einbettungsvorgang entspricht.

Alternative Befestigungsmethoden, wie die Benutzung von flüssigen Kaltklebern, finden ebenfalls Verwendung, doch als sehr zuverlässig gilt nach wie vor die Heißklebetechnik. Das größte Problem mit in die gefräste Modulaussparung getropften Flüssigklebern ist das Aushärten und die nicht exakt definierte Verbindungsfläche zwischen Modul und Kartenkörper.

Ist das Modul in den Kartenkörper implantiert und dieser mit allen personenunabhängigen Merkmalen und Beschriftungen versehen, so ist die Chipkarte mechanisch fertig gestellt.

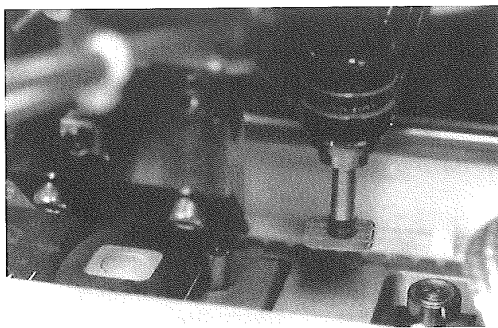


Bild 10.49 Einsetzen des Moduls in die Modulaussparung am Implanter. (Mühlbauer)

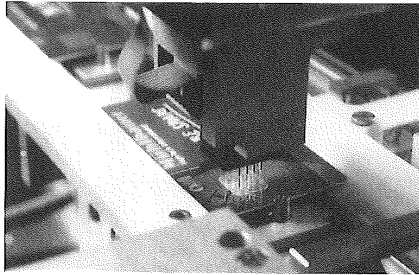


Bild 10.50 Elektrische Prüfung des in die Karte eingeklebten Moduls am Implanter. (Mühlbauer)

10.3 Phase 2 des Lebenszyklus im Detail

Die Phase 2 des Lebenszyklus von Chipkarten nach ISO 10 202-1 beschreibt neben der Implantierung der Chips in den vorbereiteten Kartenkörper auch das Laden aller Daten, die nicht kartenindividuell sind. Sowohl Phase 2 und 3 werden oft von einer einzelnen Firma ausgeführt, obwohl die beiden Phasen dort aus Sicherheitsgründen meist organisatorisch und räumlich vollständig getrennt sind.

Zur Koordination dieser komplexen Produktionsabläufe kommt häufig ein Produktionsplanungs- und Steuerungssystem (PPS-System) zum Einsatz. Von diesem System beziehen die unterschiedlichen Fertigungsmaschinen ihre Daten und melden parallel dazu den aktuellen Produktionsstatus an einen zentralen Leitstand zurück. So kann die Massenproduktion der Chipkarten zeit- und kostenminimal gesteuert werden. Dieses System hat noch den nützlichen Nebeneffekt, dass über die Anbindung aller Fertigungsmaschinen auch alle Daten für Qualitätssicherung und Tests für eine zeitnahe Auswertung zur Verfügung stehen.

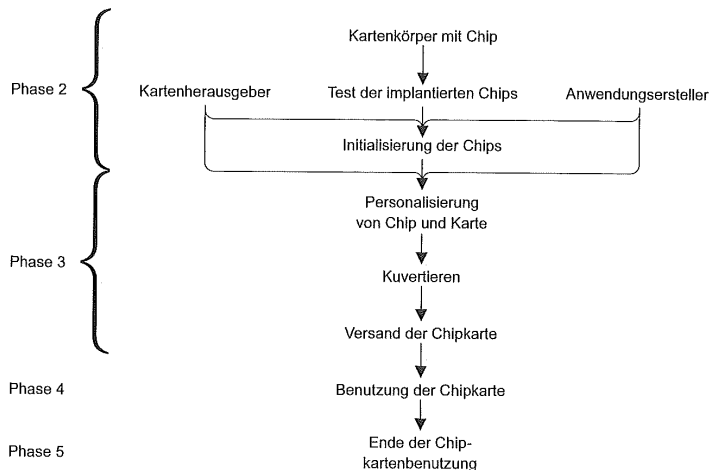


Bild 10.51 Test, Initialisierung, Personalisierung, Benutzung und Ende der Benutzung von Chipkarten in den Phasen 2 bis 5.

Datenübergabe

Der Kartenherausgeber bzw. Anwendungsanbieter muss dem Kartenpersonalisierer alle seine Anwendung betreffenden Daten mitteilen. Dazu zählen beispielsweise Name der Anwendung, Form des Dateibaums, notwendige Dateien und Dateistrukturen. Diese Daten werden bei der Initialisierung in die Chipkarte geladen. Weiterhin benötigt der Personalisierer ebenfalls alle kunden- und systemindividuellen Daten wie geheime kartenindividuelle Schlüssel, Namen und Anschrift des Karteninhabers. Die Übergabe aller dieser Daten findet durch Disketten, Magnetbänder oder per Datenfernübertragung statt.

Die Personalisierungsdaten sind in fast allen Fällen sicherheitssensibel, weshalb der Transportweg und die Übergabe entsprechend abgesichert sind. Deshalb sind sie in der Regel verschlüsselt. Den dazugehörigen Schlüssel für die Entschlüsselung transportiert man natürlich auf einem anderen Weg zum Personalisierer als die Daten. Damit sind die Personalisierungsdaten bei einem Verlust nutzlos, da eine Entschlüsselung ohne den Schlüssel nicht möglich ist.

Es existieren jedoch auch viele Chipkarten-Anwendungen, bei denen keine kartenindividuelle Datenübergabe stattfindet. Das bekannteste Beispiel dafür sind SIMs für das GSM-Mobilfunksystem, die bei der Herstellung nicht für einen bestimmten Kartenhersteller gemacht werden, sondern nur individuelle Daten und Schlüssel erhalten. Meist werden die dafür benötigten Datensätze direkt beim Kartenhersteller generiert und dann an den An-

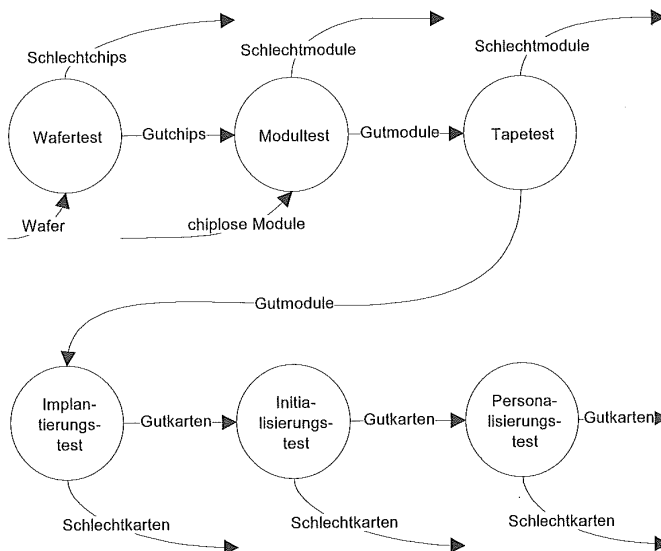


Bild 10.52 Materialflussdiagramm mit den typischen elektrischen Tests von Chipkarten-Mikrocontrollern während der Fertigung.

Wafertest: Test beim Halbleiterhersteller auf dem Wafer

Modultest: Test beim Moduleinbeter nach dem Bonden der Chips

Tapetest: Test der Module auf dem 35 mm Tape

Implantierungstest: Test der Chipkarten nach dem Implantieren der Module

Initialisierungstest: Test der Chipkarten nach der Initialisierung

Personalisierungstest: Test der Chipkarten nach der Personalisierung

wendungsbetreiber rückgemeldet, sodass dieser weiß, welche Karten produziert worden sind. Eine Datenübergabe findet dann nur in dem Sinne statt, dass der Kartenhersteller erstens die für alle Karten gleichen Daten erhält und zweitens die Start- und Endwerte für die kartenindividuellen Daten. Die Erzeugung der Datensätze für die jeweils zu produzierende Karte wird dann in Sicherheitsmodulen in der Fertigung vorgenommen.

Elektrischer Test

Der erste Fertigungsschritt dieser Phase ist ein elektrischer Test der Chipkarte. Als Basis-test wird die ISO-Anschaltsequenz für die Chipkarte gefahren, welche daraufhin einen gültigen ATR aussenden muss. Kann der ATR empfangen werden und entspricht er dem Erwarteten, so ist sichergestellt, dass zumindest der Kern des Mikrocontrollers funktioniert. Danach folgen spezielle Tests der Hardwarekomponenten wie ROM, EEPROM und RAM. Um bei diesen, zum Teil mehrere Sekunden dauernden Tests einen hohen Durchsatz zu erreichen, kommen spezielle Maschinen mit der Fähigkeit der Parallelverarbeitung von Karten zum Einsatz. Typisch sind dafür Maschinen mit einem Durchsatz von bis zu 6000 Chipkarten pro Stunde.

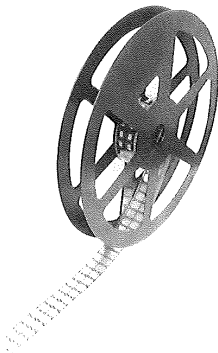


Bild 10.53 Spule mit aufgewickeltem Tape mit paarig darauf angeordneten Modulen. Dies ist der typische Lieferzustand vom Modulhersteller an der Chipkartenhersteller.

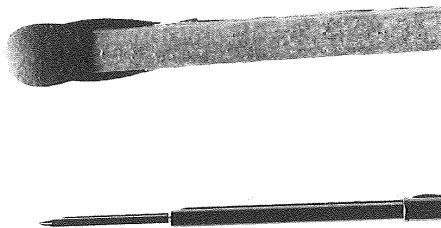


Bild 10.54 Detailansicht einer Kontaktnadel der Kontaktiereinheit für eine Hochleistungsanlage zum Testen von Chipkarten. Obwohl die Form der Kontaktnadelspitze nicht ISO/IEC 7816-2 entspricht, wird sie aus Zuverlässigkeitsgründen häufig im Fertigungsbereich eingesetzt.

Um die Funktion des EEPROMs zu testen, schreibt man vorzugsweise ein „Schachbrettmuster“, z. B. 'AA' = °1010 1010° oder '55' = °0101 0101°, in die einzelnen Bytes. Da dies aber gerade bei großen EEPROMs sehr lange dauern würde, benutzt man manchmal einen Trick, um die Testzeit zu verkürzen. Statt der vorgeschriebenen EEPROM-Schreibzeit (z. B. 3,5 ms pro Page) reduziert man diese beispielsweise um das Zehnfache auf 350 µs pro Page. Der Datenerhalt im EEPROM betrüge bei einer so kurzen Schreibdauer nur mehr wenige Minuten, doch kann man das in diesem Fall ohne Probleme in Kauf nehmen, da die Prüfung auf das Schachbrettmuster schon Sekunden nach dem Schreibvorgang abgeschlossen ist. Der Vorteil der dynamischen EEPROM-Programmierung ist eine deutlich reduzierte Prüfdauer bei gleicher Prüfqualität. Sie wird im Übrigen manchmal auch benutzt, wenn sich Empfangs- und Sendepuffer des I/O-Managers nicht im RAM, sondern im EEPROM befinden. Die Absenkung der Schreibzeit bewirkt in diesem Fall eine merkliche Steigerung der effektiven Übertragungsgeschwindigkeit.

Bei den elektrischen Tests gibt es noch einen weiteren interessanten Kunstgriff. Um den Zeitbedarf beim Laden von Daten in den nachfolgenden Fertigungsschritten zu reduzieren, schreibt man als letztes Testmuster beispielsweise '00' in den gesamten EEPROM-Speicher. Allerdings mit der regulären Schreibzeit des EEPROMs! Da man nun bei der nachfolgenden Komplettierung, Initialisierung und Personalisierung weiß, auf welchen Wert das EEPROM gesetzt ist, müssen nur noch Daten geschrieben werden, die sich von dem vorgegebenen Wert unterscheiden. Eine analoge Vorgehensweise ist auch möglich, um das EEPROM auf einen Wert zu setzen, der bei den nachfolgenden Schreibvorgängen keinen zusätzlichen Löschvorgang der jeweils zu beschreibenden EEPROM-Page notwendig macht. Beide Tricks führen zu deutlich verminderten Zeiten in allen folgenden Fertigungsschritten, bei denen Daten ins EEPROM geschrieben werden.

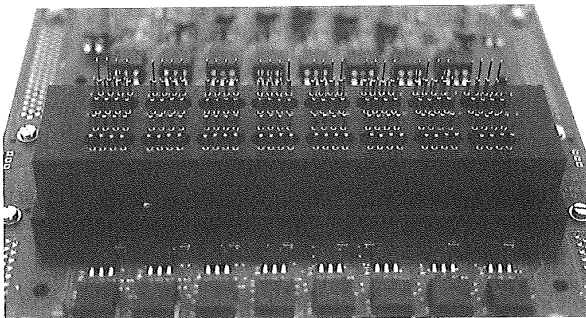


Bild 10.55 Kontaktkopf eines 16-fach parallel arbeitenden Wareneingangstesters für Module auf 35 mm Tape.

Komplettierung

Die meisten Betriebssysteme befinden sich nur in Teilen im maskenprogrammierten ROM der Chipkarte. Linktabellen und teilweise auch Programmcode werden erst nach einer Authentisierung mit einem geheimen Schlüssel in das EEPROM der Karte geladen. Man bezeichnet dieses Laden der EEPROM-Teile als das Komplettieren des Betriebssystems.⁹

⁹ Siehe auch Abschnitt 5.4 „Komplettierung“.

Diese Vorgehensweise ermöglicht kleinere Anpassungen des ROM-Programmcodes für Fehlerbeseitigungen oder spezielle Anwendungsfälle, ohne dass eine neue ROM-Maske nötig ist. Erst nachdem die Betriebssystemdaten ins EEPROM geschrieben sind, ist das Betriebssystem der Chipkarte vollständig, und es lassen sich alle Anwendungskommandos, wie SELECT und READ RECORD, ausführen.

Die Komplettierung, deren Daten bei allen Chipkarten einer Anwendung identisch sind, findet wie die Eingangstests auf parallel arbeitenden Maschinen mit sehr hohem Durchsatz statt.

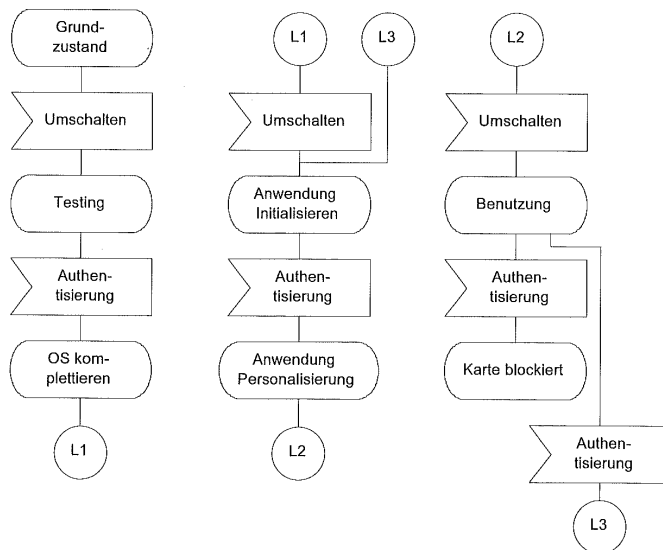


Bild 10.56 Der Zustandsautomat eines Chipkarten-Betriebssystems zur Realisation eines 5-phasigen Lebenszyklus.

Initialisierung

Die Komplettierung ist die softwaretechnische Voraussetzung, um im nächsten Fertigungsschritt alle Daten zu laden, die zu einer Anwendung gehören und in allen Chipkarten gleich sind. Dies sind zum einen alle Daten der Anwendung, die sich von Karte zu Karte nicht ändern, und zum anderen auch alle personenunabhängigen Daten, die ebenfalls in jeder Chipkarte gleich sind. Diesen Schritt bezeichnet man als Initialisierung.

Von der Dateiebene aus betrachtet, erzeugt man bei der Initialisierung alle notwendigen Dateien (MF, DFs und EFs) und füllt diese so weit wie möglich mit den Anwendungsdaten. Die Daten für die Dateiinhalte sind dabei oft schon durch die entsprechenden Spezifikationen (z. B. GSM 11.11) vorgegeben. Zur Initialisierung benutzt man in modernen Betriebssystemen die Kommandos CREATE, UPDATE BINARY und UPDATE RECORD. Dieser Fertigungsschritt ist der Letzte, in dem alle Karten gleich behandelt werden. Deshalb kann die Initialisierung auch auf schnellen und parallel arbeitenden Maschinen erfol-

gen. Die kartenindividuellen Daten der Anwendung und die personenbezogenen Daten lädt man erst im folgenden Fertigungsschritt, der Personalisierung, in die Chipkarte.

Der Grund für diese fertigungstechnische Unterscheidung in globale, allgemeine und individuelle, personenbezogene Daten, liegt in der Prozesskostenminimierung. Personalisierungsmaschinen, die fähig sind, in jede Chipkarte individuelle Daten unter den geforderten Sicherheitsbedingungen zu schreiben, sind technisch aufwendig und haben einen Durchsatz von etwa 700 Stück/h. Meist sind sie auch noch mit langsamen Beschriftungseinheiten für den Kartenkörper ausgerüstet. Dies führt zu hohen Stückkosten für das Laden der Daten. Deshalb versucht man immer, alle globalen Daten, die nicht kartenindividuell sind, in den einfacheren und schnelleren (etwa 3 500 Stück/h) Maschinen für die Initialisierung in die Chipkarte einzubringen.

Der Flaschenhals sowohl bei der Initialisierung als auch bei der Personalisierung ist die Übertragung und das anschließende Schreiben der Daten in das EEPROM. Die Schreibzugriffe auf das EEPROM können aufgrund technischer Grenzen momentan nicht reduziert werden. Die Zeit für die Übertragung der Initialisierungs- und Personalisierungsdaten kann aber durch Erhöhung der Taktfrequenz und Reduzierung des Teilers der Datenübertragung stark gesenkt werden. So arbeiten manche Initialisierungs- und Personalisierungsmaschinen mit Datenübertragungsraten bis zu 115 kBit/s statt den in der Chipkartentechnik üblichen 9 600 Bit/s. Das Ergebnis ist eine zum Teil auf mehr als die Hälfte reduzierte Zeit für die Initialisierung bzw. die Personalisierung.

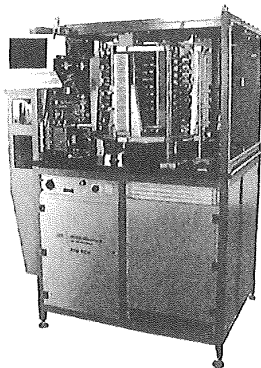


Bild 10.57 Ansicht einer 40-fach parallel arbeitenden Initialisierungsmaschine. (Mühlbauer)

Das folgende Rechenbeispiel soll verdeutlichen, dass sich bei der Massenproduktion von Chipkarten selbst geringste Zeitoptimierungen lohnen. Die folgenden Annahmen sollen dabei gelten: Es sind 1 Million Chipkarten mit 4 kByte Daten (= 4096 Byte) zu initialisieren, wobei 2 Initialisierungsmaschinen im 2-Schicht-Betrieb (16 Stunden/Tag) zur Verfügung stehen. Die Initialisierung wird mit 40 Kommandos durchgeführt, wobei das Übertragungsprotokoll $T = 1$ mit 12 Bit zu übertragenden Daten pro Byte benutzt wird. Weiterhin gilt eine EEPROM-Schreibzeit von 3,5 ms pro 4 Byte Page, wobei ein vorheriger Löschvorgang nicht notwendig ist. Die Transportzeit auf den nicht mit parallel arbeitenden Terminals ausgestatteten Initialisierungsmaschinen beträgt 1 Sekunde pro Chipkarte, und

die eventuell anfallenden Rüstzeiten (z. B.: für Magazinierung o. Ä.) werden nicht berücksichtigt. Die daraus resultierende Taktzeit ist somit die Summe aus EEPROM-Schreibzeit, Übertragungszeit und Transportzeit.

Mit den Formeln aus Abschnitt 14.2 (Formelsammlung zur Abschätzung von Ausführungszeiten) ergibt sich bei einer Datenübertragungsrate von 9,6 kBit/s eine Produktionsdauer von 90,7 Tagen. Bei einer Erhöhung der Datenübertragungsrate auf 38,4 kBit/s reduziert sich die Zeit für die Produktion der einen Million Karten auf 52,5 Tage. Ideal wäre eine Übertragungsrate von 115 kBit/s, denn dann wäre man gegenüber 9600 Bit/s mehr als 46 Tage früher mit der Produktion der Karten fertig.

Tabelle 10.4 Die Produktionsdauer bei der Initialisierung von Chipkarten in Abhängigkeit von der Datenübertragungsgeschwindigkeit. Die zugrunde gelegten Rahmenbedingungen sind im Text erläutert.

Datenübertragungsgeschwindigkeit	9 600 Bit/s	38 400 Bit/s	115 200 Bit/s
EEPROM-Schreibzeit	3 584 ms	3 584 ms	3 584 ms
Übertragungszeit	5 870 ms	1 468 ms	489 ms
resultierende Taktzeit	10 454 ms	6 051 ms	5 073 ms
resultierende Produktionsdauer	90,7 Tage	52,5 Tage	44,0 Tage

Man sieht anhand obiger Rechenbeispiele deutlich, dass es sich gerade bei der Speicherung von großen Datenmengen in Chipkarten lohnt, Zeit in Optimierungen zu stecken. Die beschriebene Erhöhung der Übertragungsrate ist im Prinzip nur vom Chipkarten-Betriebssystem abhängig und erfordert keine besondere Chiphardware, wie sie etwa ein schnelleres Schreiben von Daten ins EEPROM verlangen würde. Deshalb ist diese Senkung der Initialisierungszeit auch bei allen entsprechend vorbereiteten Chipkarten möglich.

10.4 Phase 3 des Lebenszyklus im Detail

Die Phase 3 deckt hauptsächlich den Teil der optischen und elektrischen Personalisierung der Chipkarte ab. Analog zur Phase 2 findet dieser Arbeitsgang üblicherweise in einer hochautomatisierten Fertigungsumgebung statt, die auf große Stückzahlen ausgelegt ist.

Erzeugung von kartenindividuellen Geheimnissen

Im Regelfall werden die individuellen Daten für die Personalisierung vom Kartenherausgeber auf Datenträger oder per DFÜ bereitgestellt. Im Falle geheimer Daten wie PINs oder Schlüsseln wird jedoch oft ein Sonderweg eingeschlagen, da diese Daten unter allen Umständen geheim bleiben müssen und die Generierung nur in hochsicherer Umgebung durchgeführt werden darf. Bei PINs existieren dazu vier praxisrelevante Verfahren.

Die einfachste Variante ist die Erzeugung einer Trivial-PIN, die bei der ersten Verwendung der Chipkarte und noch vor der ersten gültigen Transaktion vom Kartenbenutzer in eine PIN seiner Wahl geändert werden muss. Diese Variante lässt sich aber aus vielerlei Gründen nicht bei allen Systemen in die Tat umsetzen, obwohl bei ihr vorteilhaft ist, dass keine PIN-Briefe gedruckt und verschickt werden müssen.

Eine etwas aufwendigere Variante zur Erzeugung von PINs ist die Generierung beim Kartenherausgeber mit einem guten Zufallsgenerator und eine anschließende sichere Übergabe der PINs an den Kartenpersonalisierer. Dieser schreibt sie dann mit den üblichen sicheren Mechanismen in die zu personalisierenden Chipkarten und erstellt die dazugehörigen PIN-Briefe. Eine Untervariante davon ist die Erzeugung der PINs auf der Karte und die darauf folgende gesicherte Übertragung zwecks Weiterverarbeitung an die Personalisierungsmaschine.

Die dritte Möglichkeit ist die Erzeugung von Zufalls-PINs beim Kartenpersonalisierer. Die in sicherer Umgebung generierten PINs werden wie in der obigen Variante in die entsprechenden Datenfelder der Chipkarten geschrieben. Parallel dazu wird ein PIN-Brief erstellt, der an den Kartenbenutzer verschickt wird. Die dazugehörige Chipkarte gelangt auf getrenntem Weg zum Kartennutzer. Falls der Kartenherausgeber die erzeugten PINs benötigt, ist es möglich, ihm diese in sicherer Weise zukommen zu lassen. Ansonsten ist es meist nicht nötig, die erzeugten PINs noch zusätzlich an einem anderen Ort als in der Chipkarte zu speichern.

Ein weiterer Weg zur Erzeugung von PINs ist die Verwendung eines (kryptografischen) Algorithmus, mit dem aus auf der Chipkarte befindlichen Daten und einem Hauptschlüssel eine kartenindividuelle PIN errechnet werden kann. Nachteilig ist bei diesem Verfahren, dass der Hauptschlüssel und manchmal auch der Algorithmus geheim gehalten werden müssen.¹⁰

Handelt es sich bei den zu erzeugenden Geheimnissen nicht um PINs, sondern um Schlüssel für Kryptoalgorithmen, können grundsätzlich ähnliche Methoden angewandt werden. Der Hauptunterschied besteht darin, dass hierbei kein PIN-Brief erzeugt werden muss, sondern die Schlüssel in sicherer Weise dem Systembetreiber zur Verfügung zu stellen sind. Dies geschieht über die so genannten Antwortdaten (*response data*), die von der Erzeugungsinstanz via DFÜ oder physischen Datenträger in kryptografisch gesicherter Form dem Systembetreiber übergeben werden.

Datentransfer in die Chipkarte

Es gibt zwei grundlegend unterschiedliche Vorgehensweisen, um die Daten für die Initialisierung in den Speicher des Mikrocontrollers einzubringen. Die Erste orientiert sich an der Vermeidung von direkter physikalischer Adressierung des Speichers. Dabei werden so weit wie möglich bei Initialisierung und Personalisierung nur logische Adressen im Mikrocontroller verwendet. Vom Standpunkt der reinen Lehre aus gesehen, ist dies die zu präferierende Vorgehensweise, da damit außerhalb der Chipkarte nicht mit physikalischen Adressen gearbeitet werden muss. Dies schließt viele Fehlerquellen von vornherein aus und macht auch in gewissem Rahmen das Einbringen von Daten in die Chipkarte unabhängig vom jeweiligen Typ des Mikrocontrollers. Der Nachteil dieses Weges ist jedoch ein deutlich erhöhter Zeitbedarf für Initialisierung und Personalisierung. Zeit ist jedoch gerade bei einer Massenproduktion ein sehr kritischer Punkt.

Als Konsequenz existiert in der Praxis noch ein zweiter Weg, um Daten in eine Chipkarte einzubringen. Dabei werden die zu initialisierenden Daten von außen direkt auf phy-

¹⁰ Ein Beispiel zu diesem PIN-Erzeugungsverfahren finden Sie in Abschnitt 8.1.1 „Prüfung einer Geheimzahl“.

sikalische Adressen im Speicher des Mikrocontrollers geschrieben. Dies reduziert gegenüber dem an logischen Adressen orientierten Vorgehen deutlich den dafür notwendigen Zeitbedarf. Leider bewirkt dies auch die Erfordernis, außerhalb der Chipkarte mit physikalischen Adressen zu arbeiten, was entsprechende Nachteile hinsichtlich Fehleranfälligkeit und universeller Einsetzbarkeit zur Folge hat. In der Praxis wird die Vorgehensweise in der Regel von Fall zu Fall entschieden. Erreichen die zu produzierenden Chipkarten entsprechend große Stückzahlen, lohnt sich der erhöhte Aufwand für die Software der Initialisierungsmaschinen und des notwendigen diffizilen Testes.

Um Daten direkt auf physikalische Adressen zu schreiben, müssen sie einleitend entsprechend aufbereitet werden. Dazu müsste die gesamte Dateiverwaltung des Chipkarten-Betriebssystems als Simulation nachgebildet werden. Man könnte dann mit einem Konvertierungsprogramm die zu schreibenden Daten in entsprechend codierte Dateibodies der Simulation einbringen und mit den dazugehörigen Dateihedern versehen. Die in dieser Weise aufgebauten Dateien müssten dann nur noch an die richtige Adresse im Speicher relokiiert werden. Der ganze Vorgang muss selbstverständlich fehlerfrei und auf die jeweilige Betriebssystemversion abgestimmt durchgeführt werden. Anschließend könnte man die

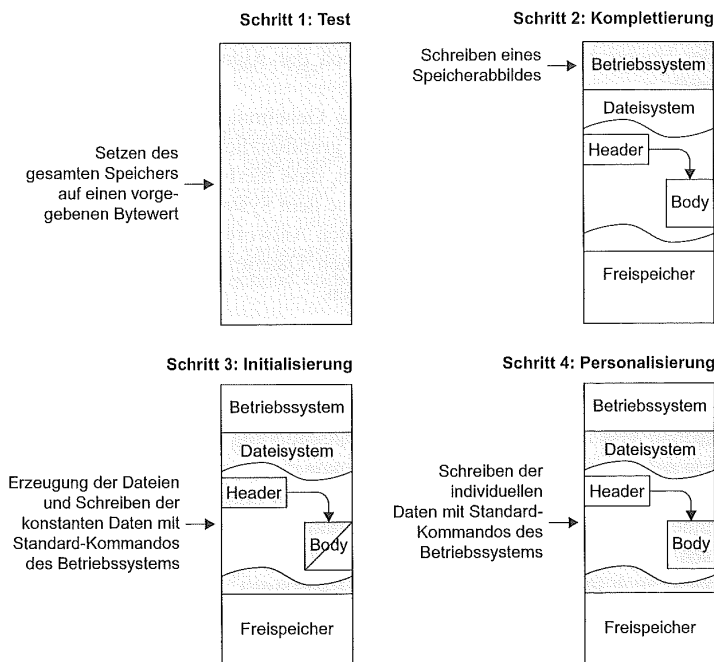


Bild 10.58 Schematische Darstellung der wichtigsten Schritte bei der Einbringung der einheitlichen und individuellen Daten in einer Chipkarte durch logische Kommandos wie beispielsweise CREATE, UPDATE BINARY und UPDATE RECORD. Die grau hinterlegten Bereiche kennzeichnen in dem jeweiligen Schritt geschriebene Daten. Aus Vereinfachungsgründen ist hier lediglich das Laden eines Dateisystems dargestellt, analoge Schritte würden beispielsweise auch beim Laden von Java-Applets durchgeführt werden.

Daten aus der Simulation auslesen und mit den üblichen Kommandos direkt auf physikalische Speicheradressen in die Chipkarte schreiben. Leider hat sich dieser Weg in der Praxis nicht bewährt, da der nötige Aufwand den Nutzen bei weitem übersteigt. Zudem wäre der Testaufwand für die aufwendigen Black-Box-Tests zur Sicherstellung der Fehlerfreiheit zu hoch. Deshalb wird diese Vorgehensweise selten eingesetzt.

Der übliche Weg ist viel einfacher. Eine Chipkarte, die in einem ansonsten nicht benutzten Speicherbereich ein Dump-Programm enthält, wird mittels logischer Kommandos initialisiert. Mit dem Dump-Programm werden anschließend der initialisierte Speicher ausgelesen und die erhaltenen Daten auf die physikalischen Adressen der zu initialisierenden Chipkarten geschrieben. Damit lässt sich eine Reduktion der Initialisierungs- und Personalisierungszeiten um bis zu 30 % erreichen. Im Prinzip könnte man diesen Vorgang auch als Klonen bezeichnen. Der große Vorteil ist die Einfachheit und Robustheit. Kritisch ist nur,

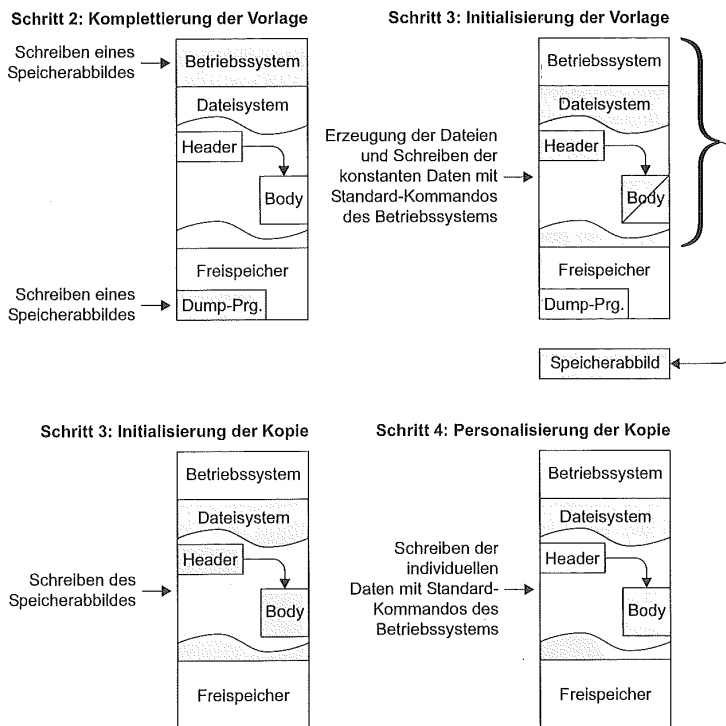


Bild 10.59 Schematische Darstellung der wichtigsten Schritte bei der Einbringung der einheitlichen und individuellen Daten in einer Chipkarte durch das physikalische Kopieren von Daten aus einer zuvor mittels logischer Kommandos erstellten Vorlage. Zusätzliche Details der dazugehörigen Abläufe für Vorlage und Kopie sind noch in Bild 10.58 dargestellt, von dem auch die Systematik der Nummerierung übernommen wurde. Die grau hinterlegten Bereiche kennzeichnen in dem jeweiligen Schritt geschriebene Daten. Aus Vereinfachungsgründen ist hier lediglich das Laden eines Dateisystems dargestellt, analog Schritte würden beispielsweise auch beim Laden von Java Applets durchgeführt werden.