Die approbierte Originalversion dieser Diplom-/Masterarbeit ist an der Hauptbibliothek der Technischen Universität Wien aufgestellt (http://www.ub.tuwien.ac.at).

The approved original version of this diploma or master thesis is available at the main library of the Vienna University of Technology (http://www.ub.tuwien.ac.at/englweb/).

FAKULTÄT FÜR !NFORMATIK

Didaktische Unterlagen für Hardwaretoken Betriebssysteme

mit spezieller Berücksichtigung der Datenverwaltung

MASTERARBEIT

zur Erlangung des akademischen Grades

Magister der Sozial- und Wirtschaftswissenschaften

im Rahmen des Studiums

Informatikmanagement

eingereicht von

Markus Winkler
Matrikelnummer 0225272

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung:
Betreuer: Univ. Doz. Dipl.-Ing. Dr. techn. Ernst Piller

Wien, 14. 10. 2008

(Unterschrift Verfasser/in) (Unterschrift Betreuer/in)

Abstract

Content of this work is an approach to give the reader a simple introduction for self-learning to the quite complex world of operating system for hardware token, in particular IC cards. Furthermore there are slides in the appandix for use in a lecture, including an exercise course.
The first step is the illustration of the history and the development of IC cards. Next is the design of an IC card and the description of an operating system for IC cards according to ISO. Due to the fact that norms are playing an important role in the development of IC card op

erating systems, as IC cards need to be read all over the world (ATM), there is a chapter describing the norms. The focus of the description of the operating system is set on the file management as it is of huge importance. For a better illustration the file management is compared to the file management principles of the well known but different types of PC operating systems.

Zusammenfassung

In dieser Arbeit wird versucht, eine einfache, fürs Selbsstudium geeignete Einführung in das doch recht komplexe Gebiet der Betriebssysteme für Hardware Token, im besonderen Chipkarten, zu geben. Des weiteren werden im Anhang Folien bereitgestellt, wie sie in einer Vorlesung mit begleitender Übung verwendet werden können.
Zuerst wird die Entwicklung und Geschichte der Chipkarten dargestellt. Danach wird ein Einblick in den Aufbau einer Chipkarte gegeben und erläutert, wie ein Betriebssystem für Chipkarten nach ISO-Norm aufgebaut werden sollte. Aufgrund der Tatsache dass Normen eine sehr wichtige Rolle in der Entwicklung von Betriebssystemen für Chipkarten spielen, da Chipkarten überall auf der Welt an kompatiblen Geräten (z.B.: Geldautomaten) gelesen werden sollten, beschreibt ein Kapitel die wesentlichen Normen. Bei der Beschreibung zum Aufbau eines Chipkartenbetriebssystems wird im speziellen auf die Dateiverwaltung eingegangen, welche eine wichtige Rolle spielt. Hierzu wird die Dateiverwaltung von Chipkarten mit der sehr bekannten doch recht differierenden Dateiverwaltung von PC Betriebssystem verglichen.

Danksagung

An erster Stelle möchte ich meinem Vater danken, der mir das Studium und somit auch diese Arbeit erst ermöglicht hat.
Weiters bedanke ich mich bei meiner Freundin, die mich die ganze Zeit über ertragen musste. Besonderen Dank gilt auch meinem Onkel Fritz.

Weiteres bedanke ich mich herzlichst bei Herrn Univ. Doz. Dipl.-Ing. Dr. techn. Ernst Piller für die Betreuung meiner Arbeit.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einführung

1.1. Motivation

Ziel der vorliegenden Arbeit ist es, dem Leser mit den vorliegenden Unterlagen eine Einführung in das komplexe Gebiet der Hardware Token Betriebssysteme zu geben. Im Anhang finden sich Folien die verwendet werden können die behandelten Themen einem Publikum näher zu bringen. Über dieses Thema existieren nach Meinung des Autors noch recht wenige gute und leicht verständliche Werke. Aus diesem Grund wird versucht, mit dieser Arbeit ein Werk zu schaffen, welches einen einfachen Einstieg in diese komplexe Materie vermittelt. Da bei Hardware Token Betriebssystemen die Dateiverwaltung, die im Vergleich zu PC Systemen doch recht einfach ist, ein wesentliches Element darstellt, liegt der Focus dieser Arbeit im Besonderen auf diesem Gebiet der Betriebssystementwicklung für Hardware Token.

1.2. Überblick über die Arbeit

Das Kapitel 2 der vorliegenden Arbeit beschäftigt sich zuerst mit der Geschichte und Entwicklung der Chipkarten. Es wird versucht zu zeigen, wie sich aus einfachen Papierkarten eine Industrie rund um mikroprozessorgesteuerte Plastikkarten entwickelt hat. Des Weiteren werden damalige wie auch heutige Anwendungsgebiete von Chipkarten aufgezeigt. Dies umfasst den Zahlungsverkehr, wie auch Chipkarten für das Benützen eines Skiliftes in einem Wintersportgebiet. Daneben werden verschiedene Kartentypen dargestellt und nach Verwendungstyp bzw. Leistungsfähigkeit klassifiziert.

Da Normen ein wesentlicher Faktor dafür waren, daß Chipkarten sich weltweit so rasant ausbreiten konnten, wird erklärt, wer sich für die Normung bei Chipkarten verwantwortlich zeichnet und welche Normen existieren. Die Geschichte hat sehr viele verschiedene Möglichkeiten hervorgebracht, wie auf einer Plastikkarte Daten abgespeichert werden können. Der Aufdruck des Namens und einer Nummer ist nur eine Form. Weitere Formen werden beschrieben, und welche Norm diese regelt.
Das Kapitel 3 ist der Kern der Arbeit. Hier wird zuerst das von Microsoft entwickelte Dateisystem FAT im Detail beschrieben um später Analogien zu den Chipkartensystemen ziehen zu können. FAT ist ein doch recht altes Dateisystem, doch die Idee läßt sich anhand von FAT sehr gut weitergeben. Folgend werden Dateien auf Chipkarten genau beschrieben. Der Aufbau und auch die Dateitypen unterscheiden sich enorm von ihren Pendaten auf dem PC. Es wird des Weiteren beschrieben, wie Dateien in einem Betriebssystem für Chipkarten verwendet werden können. Bei Chipkarten gibt es viele unterschiedliche Arten, Dateien zu verwalten. Dies hängt hauptsächlich von der Art der Anwendung und auch der Komplexität ab. Die Verwaltung mittels FAT ist eine M̈oglichkeit unter vielen, die beschrieben wird. Weiters sind Zugriffsbedinungen wichtig, wenn Daten vor unbefugtem Zugriff geschützt werden sollten. Hier gibt es auch eine Norm, die diese Vorgehensweise genau beschreibt. Das letzte Kapitel beschäftigt sich abschließend mit Strategien für die Speicherverwaltung bei Chipkarten.
Zusätzlich befinden sich im Anhang Folien, die sich hervorragend eignen, um den vorliegenden Stoff einem Publikum näher zu bringen.

2. Überblick Chipkarten

Diese Kapitel zeigt die Geschichte und Entwicklung von Plastikkarten, im Besonderen Chipkarten angelehnt an die Ausführung von Rankl (2007).

2.1. Geschichte

Bis in die 50er Jahre des 20. Jahrhunderts waren Papierkarten verbreitet, die ähnliche Zwecke erfüllten wie die anfänglichen Plastikkarten. Da der Kunstoff PVC langlebiger als Papierkarten war und vorallem sehr preisgünstig wurde, begann die Herstellung von Plastikkarten. Zur damaligen Zeit reichte es aus, dass die Karte den Namen des Inhabers enthielt und die Unterschrift. Diners-Club gab die ersten Plastikkarten aus, jedoch waren diese nur einem sehr exklusiven Personenkreis vorbehalten. Somit ließ sich mit der Plastikkarte nicht mehr mit Geld bezahlen, man konnte von nun an mit seinem guten Namen bezahlen.
Als dann auch noch Mastercard und VISA auf den Markt aufmerksam wurden und einstiegen, verbreitete sich die Plastikkarte sehr rasch. Den Ursprung hat die Plastikkarte in Amerika doch ein paar Jahre nach der Verbreitung in Amerika, kam sie auch bald nach Europa und in den Rest der Welt.
Der Vorteil von Plastikkarten im Vergleich zu Bargeld war klar. Mit der Plastikkarte wurde das Risko drastisch verringert, dass Bargeld auf Reisen verschwindet. Ein weiterer Vorteil war es, dass mit der Platikkarte in vielen Ländern gezahlt werden konnte, ohne dass der Umtausch in eine Landeswährung erfolgen musste. Doch schon bald kamen mit der Verbreitung der Platikkarte auch die ersten Betrugsversuche. Somit entstanden auch schon die ersten Sicherheitsmerkmale von Karten, welche teilweise noch auf heutigen Chipkarten vorzufinden sind. Die ersten Maßnahmen gegen Betrug waren

nur visuelle Merkmale, wie zum Beispiel die Unterschrift und auch die Hochprägung, die noch heute bei Kreditkarten verwendet werden.
Spätere Sicherheitsmaßnahmen waren der Einsatz von Magnetstreifen auf der Karte, auf dem Daten wie zum Beispiel der Name des Inhabers gespeichert wurde, und auch der Einsatz einer PIN, die nicht auf der Karte gespeichert wurde, sondern auf einem Host Computer. Der Einsatz einer PIN brachte ein wesentlich höheres Maß an Sicherheit, da der Magnetstreifen einer Karte von jedem ausgelesen werden kann, der die notwendige Hardware dafür besitzt. Da die PIN aber nicht auf der Karte gespeichert wurde, war es so gut wie unmöglich diese durch technische Maßnahmen herauszufinden. Der Nachteil der Verwendung einer PIN war, dass jedes Terminal nun mit einem Host Computer verbunden sein musste.

Telefonwertkarten

Mit den Fortschritten der Mikroelektronik in den 70er Jahren kam dann der Chip auf die Karte. Die Chipkarte war geboren. Schon im Jahr 1968 wurde die Idee von einem Chip auf einer Plastikkarte von *Jürgen Dethloff* in Deutschland zum Patent angemeldet (Rankl (2007). Erst als 1974 Roland Moreno seine Patente zu Chipkarten in Frankreich anmeldete kam die Entwicklung langsam in Schwung. Ein Grund hierfür ist sicherlich, dass im Jahr 1968 die Technik noch nicht weit genug war um Chipkarten im Großen Maßstab konstengünstig zu fertigen.
Der Durchbruch jedoch kam 1984 als die PTT (Postes, télégraphes et téléphones) Frankreich einen großangelegten Feldversuch zur Verwendung von Telefonwertkarten durchführten. Durch den Feldversuch wurde festgestellt, dass die Karten die hohen Anforderungen an Manipulationssicherheit und Zuverlässigkeit tatsächlich erfüllten. Neben Frankreich machte auch Deutschland Versuche, unter anderem auch mit Hologrammkarten und Karten mit Magnetstreifen. Frankreich setzte auf kostengünstigere EPROMs und Deutschland auf die neueren EEPROMs, was dazu führte, dass die Chipkarten der zwei Länder zueinander nicht kompatibel waren.

Bankkarten

Die in Telefonkarten eingesetzten Chips waren von sehr einfacher Natur und somit auch kostengünstig, was zu einer sehr schnellen Verbreitung der Telefonkarten beigetragen hatte. Die Entwicklung und Verbreitung von Chipkarten im Bankenwesen lief da schon wesentlich langsamer an. Bei Bankkarten kommt es genauso wie bei Telefonkarten auf kostengünstige Herstellung an, aber zudem muss eine Bankkarte ein Maximum an Sicherheit bieten. Als die moderene Kryptographie (ein Überblick über wichtige Verfahren, auch bei Chipkarten findet sich in Buchmann (2008)) einen enormen Aufschwung durch die elektronische Datenverarbeitung erfuhr, konnten komplexe kryptographische Verfahren in sehr kurzer Zeit druchgeführt werden. Bis in die 60er Jahre des 20. Jahrhunderts war die Kryptographie dem Militär vorbehalten. Doch durch die großen Fortschritte in der Mikroelektronik änderte sich das rasch, und die Kryptographie fand auch in nicht militärischen Anwendungen Verwendung.
Auch bei den Bankkarten waren die Franzosen diejenigen, die den Startschuß abgaben. Französische Banken entschieden sich im Jahr 1984 dafür, die neue Technik der Chipkarten einzusetzen. Schon 2 Jahre zuvor wurde ein Pilotversuch gestartet, bei dem ca. 60.000 Karten ausgegeben wurden. Innerhalb der nächsten 10 Jahre wurden allmälich alle französischen Bankkarten mit Chip ausgerüstet. Das Land Österreich führte 1996 eine multifunktionale Chipkarte mit POS Funktion ein, und war somit das erste Land weltweit, welches ein flächendeckendes Geldbörsensystem auf Chipkartenbasis hatte.
Ein weiterer Meilenstein war die EMV (Europay, Mastercard, Visa) Spezifikation, die 1994 als quasi Standard veröffentlicht wurde.

2.2. Anwendungsgebiete von Chipkarten

Nach der rasanten Verbreitung der Chipkarten am Ende des 20. Jahrhunderts fanden sich auch immer mehr neue Anwendungsgebiete für Chipkarten. Einige Anwendungen wurden schon im geschichtlichen Überblick von Chipkarten erwähnt. Nach Art der Anwendung können Chipkarten grob in drei verschiedene Klassen unterteilt werden.

2.2.1. Speicherkarten

Die einfachsten Arten von Speicherkarten kamen schon als Telefonwertkarten zum Einsatz. Anfangs waren es ganz einfach Plastikkarten auf denen mit aufgedruckten Markierungen der Wert der Karte vor Gebrauch festgelegt wurde. Wenn nun die Karte in Verwendung war, wurde die Wertminderung durch weitere Aufdrucke gekennzeichnet. Abbildung 2.1 zeigt eine solche Telefonwertkarte, wie sie in Österreich in Verwendung war. Soche Systeme sind heute auch mit Chipkarten realisierbar. Hier wird der Anfangswert auf die Chipkarte gespeichert und bei jeder Verwendung verringert. Natürlich muss bei einer Chipkarte verhindert werden, dass ein Benutzer der Karte den Wert selbst manipulieren kann. Beim Beispiel Telefonkarte mit Chip muss die Chipkarte einfach sicherstellen, dass ein zuvor geschriebener Wert nicht mehr rückgängig gemacht werden kann. Der Nachteil solch eines Systems ist es, dass die Karte nach der Verwendung unbrauchbar ist.
Weitere Anwendungsmöglichkeiten für Speicherkarten sind neben Telefonkarten unter anderem Parkkarten, Karten für Kantinen oder auch Wertkarten für Schwimmbäder aber auch Versicherungskarten. In Österreich ist seit einiger Zeit (2005) die **e-card** in Verwendung (siehe ecard (2008)).

Abbildung 2.1.: Telefonwertkarte in Österreich, Ende 20. Jhd.

2.2.2. Mikroprozessorkarten

Bei den Speicherkarten beschränkt sich die Logik der Karte hauptsächlich auf das Lesen und Schreiben von Daten auf die Karte. Bei manchen Anwendungen genügt ein solcher Funktionsumfang nicht aus. Ein Beispiel hierfür sind die Bankkarten. Da es bei Bankkarten sehr auf Sicherheit ankommt, müssen dort kryptographische Verfahren

zum Einsatz kommen, welche eine gewisse Rechenleistung voraussetzen. Hierfür gibt es frei programmierbare Mikroprozessoren auf Chipkarten, welche einer Anwendung im Prinzip nur Grenzen in Bezug auf Speicher und Rechnleistung setzen. Diese Karten sind somit sehr vielseitig einsetzbar und seit den 90er Jahren des letzten Jahrhunderts auch sehr preisgünstig. Dadurch finden sich immer neue Anwendungen für diesen Typ Karte. Mikroprozessorkarten wurden für GSM vorgeschrieben um somit Mobiltelefone vom Dienstleister zu entkoppeln, was sehr zur Verbreitung von Mikroprozessorkarten beitrug. Aber auch andere Anwendunge trugen zur raschen Verbreitung von Mikroprozessorkarten bei, wie zum Beispiel die elektronische Zugriffskontrolle, die elektronische Unterschrift, die elektronische Geldbörse aber auch das PayTV.

2.2.3. Kontaktlose Karten

Dir zuvor vorgestellten Speicherkarten wie auch Mikroprozessorkarten haben einen gemeinsamen Nachteil. Zur Verwendung der Karten wird immer ein Terminal benötigt. Ohne ein Terminal sind die Karten nutzlos. Bei manchen Anwendungen kann es sehr mühsam sein, wenn man eine Karte jedesmal in ein Terminal stecken muss. Aus diesem Grund haben sich die sogenannten kontaktlosen Kartn entwickelt. Sie sind im Prinzip sowohl als Mikroprozessorkarte, wie auch als Speicherkarte erhältlich. Sie erlauben dem Anwender die Verwendung, ohne dass die Karte in ein spezielles Terminal gesteckt werden muss. Anwednungen hierfür sind auch hier die elektronische Zugangskontrolle, aber auch Schipaßsysteme haben sich durchgesetzt. Ebenso eine vollautomatische Gepäckidentifikation ist hier denkbar, da kontaktlose Karten eine Reichweite von wenigen Zentimeter bis zu einem Meter erreichen. Dies macht sie aber auch sensibel gegenüber Angriffen. Die sogenannte **Man-In-The-Middle** Attacke ist bei einer kontaktlosen Karte ohne weiteres möglich und sollte bedacht werden, was bei Zahlungsverkehr sehr kritisch sein kann.

2.3. Normen

Ein wesentlicher Grund warum sich Chipkarten weltweit so rasant verbreitet haben, ist die Tatsache, dass sich Hersteller von Chipkarten auf gewisse einheitliche Regeln

verständigt haben, was zu einer hohen Kompatibilität führt. Um zum Beispiel ein Bankkartensystem einzuführen, bedarf es weltweit gültigen Normen. Ohne solche Normen wäre es unmöglich eine in Amerika ausgestellte Bankkarte in Europa zu verwenden, was natürlich die Nutzbarkeit einer solchen Karte sehr stark einschränken würde. Aus diesem Grund sind Normen für Chipkarten wesentlich.

ISO/IEC

Die **International Organization of Standardisation** ist ein weltweiter nichtstaatlicher Verband von Standardisierungsgremien. Dieser Verband besteht aus jeweils von einem Land gesendeten Gremien. Momentan sind ca. 100 Gremien, also Länder, daran beteiligt. Die ISO hat den Auftrag weltweit Normen für Güter und auch Dienstleistungen zu vereinbaren, um die Zusammenarbeit in den verschiedenen Bereichen zu fördern. Deutschland ist zum Beispiel durch das nationale Gremium DIN in der ISO vertreten.

Entstehung einer Norm

Wenn Bedarf an einer neuen Norm besteht, wird das in der Regel von einem Industriebereich bei der nationalen Normungsorganisation (z.B.: in Deutschland die DIN) angemeldet. Diese nationale Normungsorganisation schlägt in Folge der ISO diese Anmeldung als Arbeitsthema vor. Es gibt für die verschiedensten Themen spezielle Arbeitsgruppen innerhalb der ISO, die aus Experten bestehen. Diese Arbeitsgruppen entscheiden ob das neu vorgeschlagene Arbeitsthema angenommen wird und zu einer Norm definiert werden soll. Sollte diese Arbeitsgruppe sich für eine neue Norm aussprechen, wird die Detailspezifikation zwischen den Ländern diskutiert. Das Ergebnis dieser Diskussion ist ein sogenannter **Draft International Standard (DIS)**. Als letzter Schritt wird formal über den Normentwurf abgestimmt. Wird bei dieser Abstimmung eine zwei Drittel Mehrheit der ISO Mitglieder erreicht, welche aktiv mitgearbeit haben, und 75% der an der Abstimmung teilnehmenden Mitglieder, wird der abgestimmte Text als neue ISO-Norm veröffentlicht.
Neben der ISO existieren noch andere Normungsorganisationen, die sich teilweise um

spezielle Themenbereiche kümmern. Damit die Normung nicht mehrfach erfolgt, was eigentlich den Sinn einer Normung verfehlen würde, arbeitet die ISO mit anderen Normungsorganisationen zusammen. Bei der internationalen Chipkartennormung arbeitet die ISO mit der IEC und der CEN zusammen. Die Abbildung 2.3 zeigt einen Überblick über die relevanten Arbeitsgruppen in der Chipkartennormung.

2.4. Arten von Karten

Wie bei der Geschichte der Chipkarten schon erwähnt, gibt es viele unterschiedliche Anwendungen von Karten. Die Norm ISO/IEC 7810 (2003) legt die physische Gestaltung von Chipkarten fest. Unter **'Identification Cards - Physical Characteristics'** in der Norm werden drei verschieden Größen von Karten festgelegt. Dies sind die Formate **ID-1**, **ID-2** und **ID-3**. In diesem Kapitel werden verschiedene Arten von Karten und deren Beschriftung vorgestellt.

2.4.1. Hochgeprägte Karten

Die Hochprägung der Beschriftung einer Karte ist die älteste Technik zur Sicherung einer Karte und um sie maschinenlesbar zu machen. Hier wird durch das Einstanzen der Beschriftung die Karte machinenlesbar gemacht. Durch diese Prägung lässt sich die Beschriftung einer Karte mit einem einfachen mechanischen Gerät auf ein Blatt Papier übertragen, wie es bei der Bezahlung mittels Kreditkarte noch heute ab und zu vorkommt. Abbildung 2.4 zeigt eine ID-1 Karte mit Kennzeichnung der für die Hochprägung vorgesehen Felder.

Eigenschaft	Spur 1	Spur 2	Spur 3
Anzahl der Daten	max. 79 Zeichen	max. 40 Zeichen	max. 107 Zeichen
Codierung der Daten	6 Bit alphanumerischer Code	4 Bit BCD Code	5 Bit BCD Code
Datendichte	210 Bit/Inch	75 Bit/Inch	210 Bit/Inch
schreibbar	nicht erlaubt	nicht erlaubt	erlaubt

Tabelle 2.1.: Belegung und Eigenschaften der einzelnen Spuren ein Magnetkarte nach ISO/IEC 7811-6 (2002).

2.4.2. Magnetkarten

Ein großer Nachteil der Karten mit Hochprägung ist der hohe Verbrauch von Papier. Jedesmal wenn die Karte verwendet wird, muss ein Abdruck der Hochprägung auf Papier gemacht werden. Dies führt zu einem hohen Papierverbrauch und einem hohen Verwaltungsaufwand, was wiederum sehr teuer ist. Um dies zu vermeiden, werden die Daten auf einem Magnetstreifen auf der Karte digital codiert gespeichert. Um die Daten auf dem Magnetstreifen zu lesen, wird die Karte an einem Magnetkartenleser vorbeigezogen. Somit sind die Daten mit freiem Auge nicht mehr lesbar. Aber durch das elektronische Verarbeiten der Daten fällt der hohe Papierverbrauch und somit auch hohe Verwaltungsaufwand weg, was diese Technologie gesamt billiger macht. Nachteile der Magnetstreifenkarte ist die Tatsache, dass die Daten im Prinzip von jedem lesbar, aber auch änderbar sind, der ein Magnetkartenlesegerät besitzt. Weiters ist die Speicherkapazität des Magnetstreifens sehr begrenzt auf schlanke 1.000 Bit. Abbildung 2.5 zeigt die Position des Magnetstreifens und Tabelle 2.1 zeigt die Belegung bzw. die Eigenschaften der Spuren des Magnetstreifens nach ISO/IEC 7811-6 (2002).

2.4.3. Chipkarten

Die Chipkarte ist die neueste Entwicklung in der Familie der Karten im ID-1 Format. Sie besitzt einen Mikrochip und ist somit die vielseitigste der genannten Karten. Ihre Speicherkapazität ist um ein wesentliches größer als bei Magnetkarten und wird von Chipgeneration zu Chipgeneration noch größer. Ein wesentlicher Unterschied, abgesehen von dem wesentlich größeren Speicher der Chipkarte, ist die Tatsache, dass die

Chipkarte nicht nur Daten speichern kann, sie kann auch Berechnungen durchführen und ist somit die schlauste der genannten Karten.
Dadurch, dass die Chipkarte auch Berechnungen durchführen kann, besteht dadurch eine neue und viel sichere Methoden die Daten vor unbefugten Zugriffen zu schützen. Der Chip der Karte kann also alle Zugriffe auf die gespeicherten Daten kontrollieren. Die Eigenschaften von Chipkarten werden durch die Normen ISO/IEC 7816-1 (2003),ISO/IEC 7816-2 (2004), ISO/IEC 7816-3 (2002), ISO/IEC 7816-4 (2005), ISO/IEC 7816-5 (2004), ISO/IEC 7816-6 (2004), ISO/IEC 7816-7 (1999), ISO/IEC 7816-8 (2004), ISO/IEC 7816-9 (2004) und ISO/IEC 7816-10 (1999), festgelegt. Aufgrund von verschiedenem Funktionsumfang und somit auch dem differierenden Preis je nach Funktionsumfang werden Chipkarten in mehrere Kategorien eingeteilt. Abbildung 2.6 zeigt die Einteilung der Chipkarten je nach Funktionsumfang.

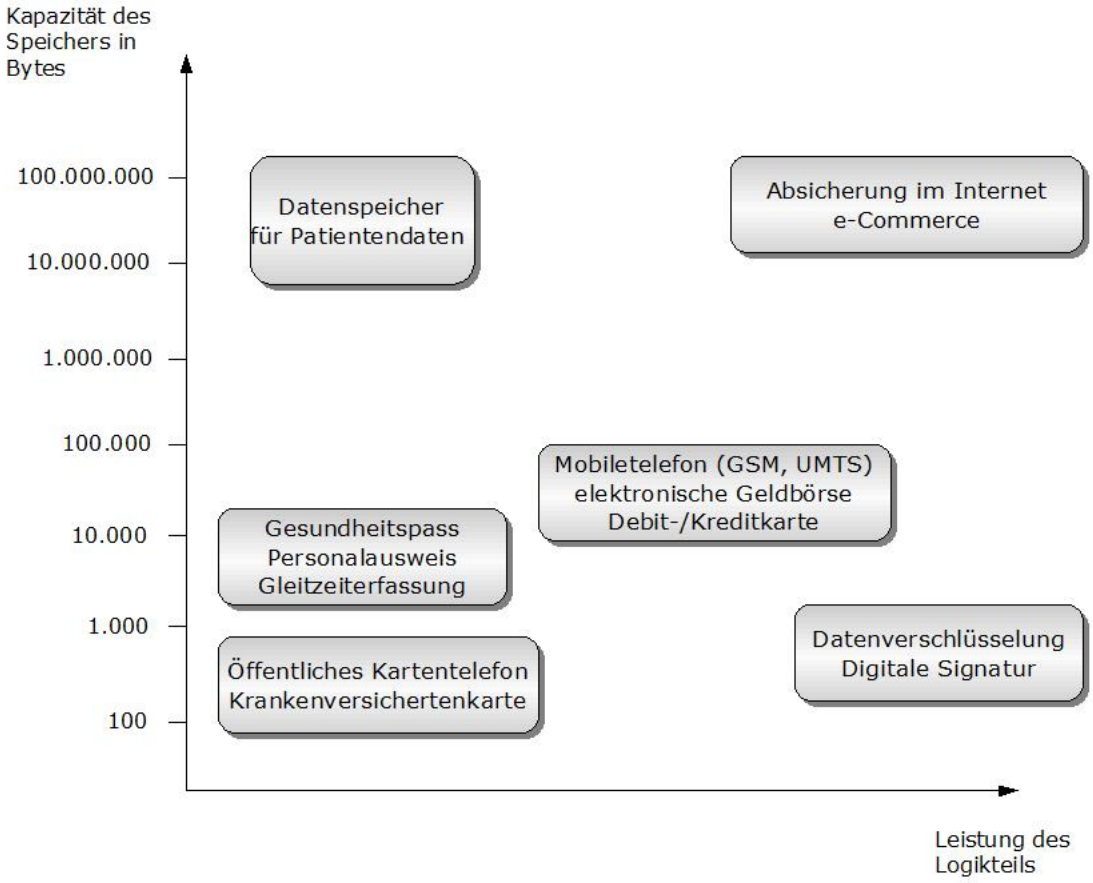


Abbildung 2.2.: Anwendungsbeispiele von Chipkarten gegliedert nach Rechneleistung und Speicherkapazität (nach Rankl (2007)).

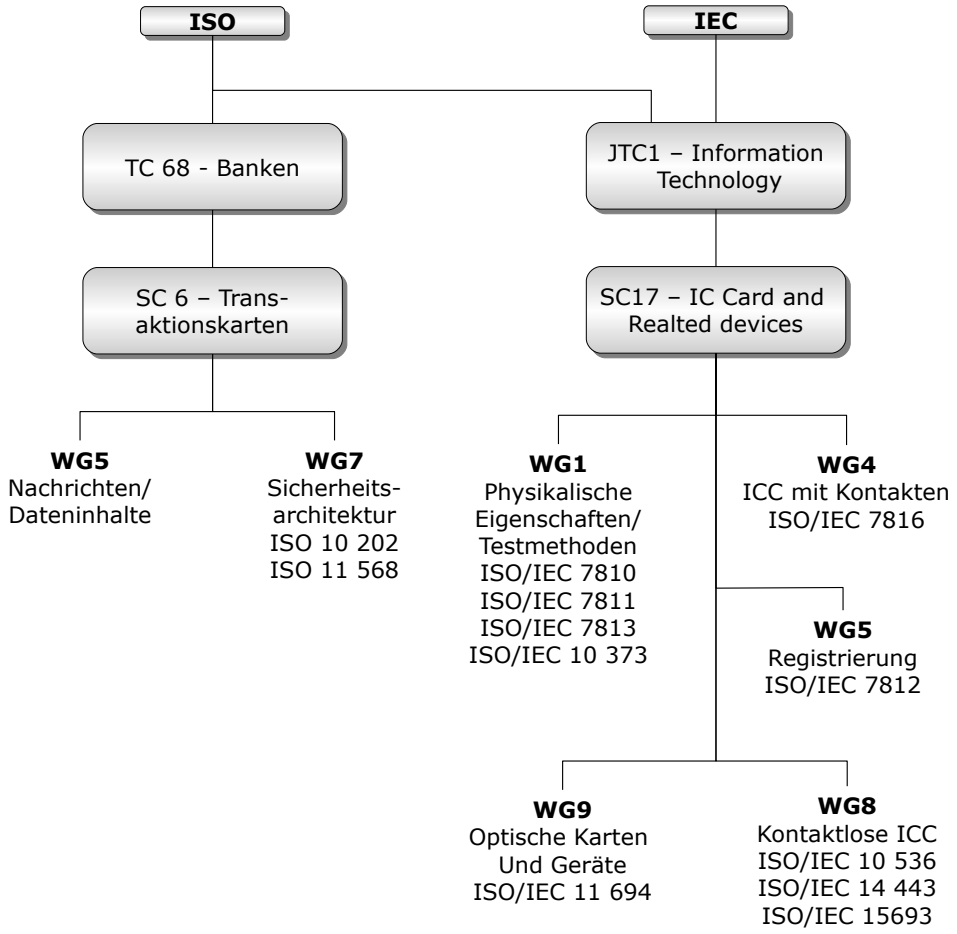


Abbildung 2.3.: Überblick über die Arbeitsgruppen bei der internationalen Chipkartennormung.

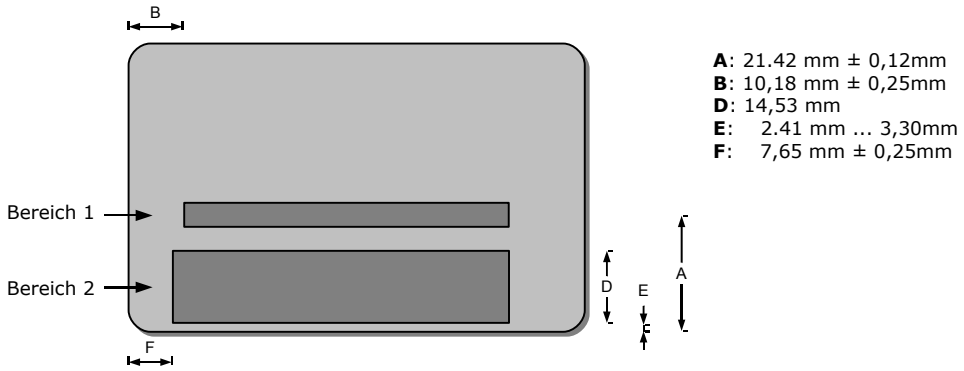


Abbildung 2.4.: Lage der Hochprägung auf einer ID-1 Karte nach ISO/IEC 7811-1 (2002).

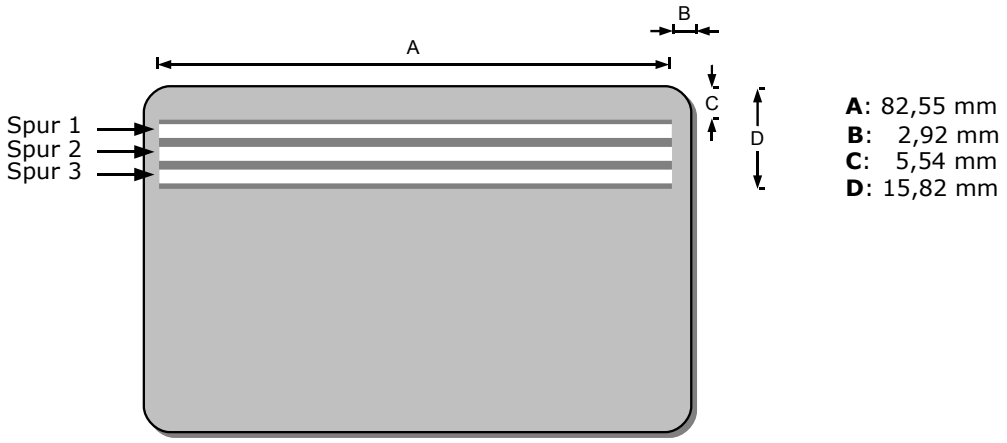


Abbildung 2.5.: Lage des Magnetstreifen auf einer ID-1 Karte nach ISO/IEC 7811-6 (2002).

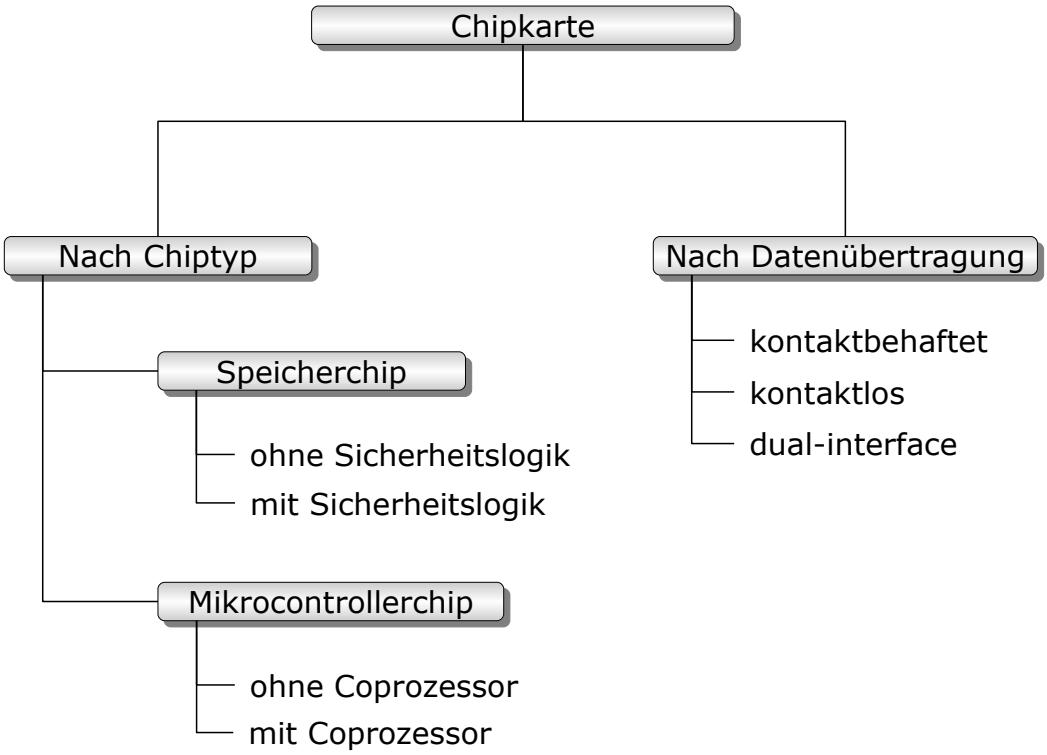


Abbildung 2.6.: verschiedene Chipkarten nach Funktion in Kategorien eingeteilt.

3. Datenverwaltung

Heutige Chipkarten erfüllen neben der Identifizierung und Authentifikation noch zusätzliche Aufgaben. Unter anderem können Chipkarten auch als Datenspeicher genützt werden. In der Praxis ersetzen sie zwar keine USB-Sticks, externe Festplatten oder ähnliches, da sich ihre Speicherkapazität im Vergleich zu den genannten externen Speichermöglichkeiten eher als bescheiden beschreiben lässt.

Dennoch bieten moderne Chipkarten die Möglichkeit die zu speichernden Daten in einem Dateisystem zu organisieren. Im folgenden Kapitel wird anhand von Microsoft's FAT erläutert, wie ein Dateiverwaltungssystem für einen PC realisiert werden kann. Im Anschluss folgt der Vergleich mit der Dateiverwaltung für Chipkarten.

3.1. Dateiverwaltung von PC Betriebssystemen

Die Dateiverwaltung von modernen Betriebssystemen ist sehr umfangreich, für einen Vergleich des Konzeptes mit dem von Betriebssystemen für Hardware Token reicht es jedoch, die grundlegende Struktur zu verstehen. Aus diesem Grund wird auf Implementierungsdetails verzichtet. Stallings (2002) beschreibt diese Details vom Konzept bis zur Implementierung genauer.

Nahezu jedes heute in Verwendung befindliche Betreibssystem für Personal Computer unterscheidet zwischen **Dateien** und **Verzeichnissen**.
Eine **Datei** beinhaltet die gespeicherten Daten, wie z.B. ein Brief, ein Bild oder auch sonstige Datensätze. Ein **Verzeichnis** hingegen beinhaltet ein oder mehrere Dateien oder auch wiederum weitere Verzeichnisse. Somit dient ein Verzeichnis im Regelfall

nicht wie eine Datei als Datenkontainer sondern mehr zur Strukturierung. Man kann sich die Struktur als eine Art auf den Kopf gestellten Baum vorstellen, an dessen Spitze sich das sogenannte *Wurzelverzeichnis* befindet.

Abbildung 3.1 zeigt eine Verzeichnisstruktur, dargestellt als Baum.

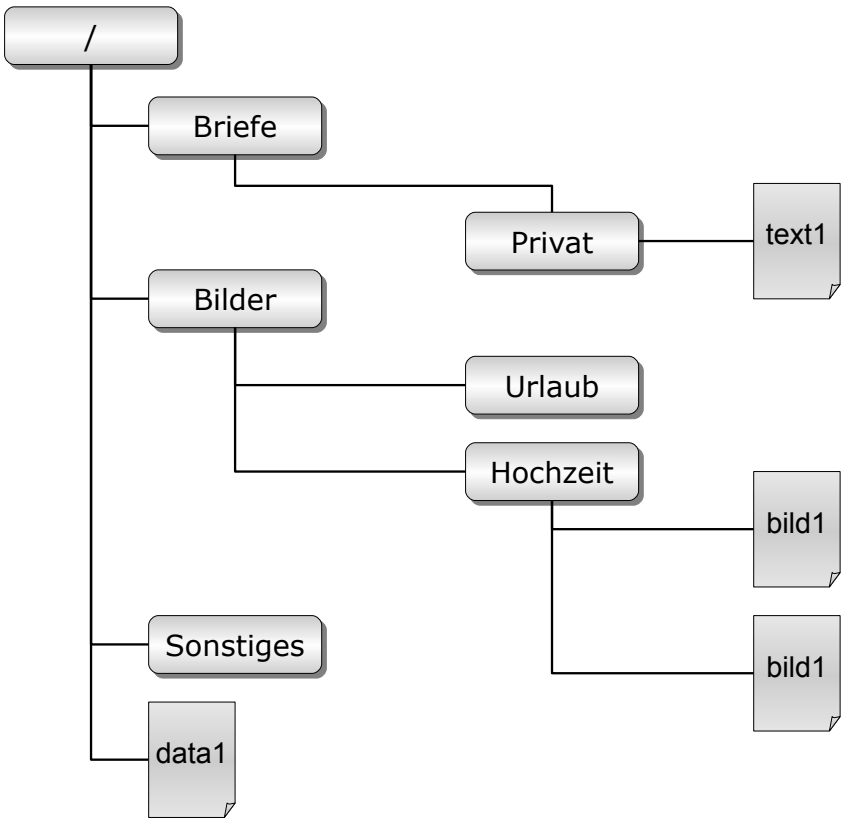


Abbildung 3.1.: Verzeichnis mit Baumstruktur.

Dies ist nur die grundlegende Logik wie Dateien durch Verzeichnisse strukturiert werden. Wie nun die Dateiorganisation von Betriebssystemen intern aufgebaut wird, entscheidet meist jedes Betriebssystem selbst. Das Betriebssystem hat direkten Zugriff auf die Festplatten eines Systems. Da die Festplatten im Prinzip keine Struktur von Haus aus aufweisen, lediglich eine lineare Anordnung von Bytes, muss auch hier das Betriebssystem eine Einteilung in für den Aufbau eines Dateisystems passende logische

Einheiten vornehmen. Der hardwaremäßige Zugriff auf die Bytes auf einer Festplatte erfolgt durch Sektoren. Die Sektorgröße ist abhängig von der Hardware, jedoch in Regel 512 Bytes. Wenn nun das 650ste Byte gelesen werden soll, müssen die ganzen 512 Bytes des 2. Sektors gelesen werden.

Das Unternehmen Microsoft hat in den letzten Jahrzehnten zwei wesentliche Dateisysteme für Festplatten eines PCs entwickelt. Das aktuelle Dateisystem, welches auf den Namen NTFS hört, ist ein modernes und auch sicheres Dateisystem, das aber schon sehr komplex geworden ist. Das ursprüngliche Dateisystem FAT, welches unter dem Betriebssystem DOS das Licht der Welt erblickte, ist um einiges einfacher aufgebaut bietet aber auch nicht die Möglichkeiten wie NTFS, vor allem im Bereich Sicherheit.

3.1.1. FAT12 und FAT16

Das FAT Dateisystem ist vom Aufbau noch recht einfach, bietet jedoch keine Rechteverwaltung. Es ist für ein System ausgelegt, an dem nur eine Person arbeitet, die auch vollen Zugriff auf das Dateisystem haben darf. Selbst auf Systemdaten des Betriebssystems hat der Benutzer vollen Zugriff. Trotz der fehlenden Rechteverwaltung wird das Dateisystem FAT auch heute noch oft verwendet, hauptsächlich auf mobilen Datenträgern, wie Disketten oder USB-Sticks.

Eine mit FAT formatierte Festplatte ist in mehrere Abschnitte unterteilt. Der erste Sektor ist der Startsektor. Danach folgt die Dateizuordnungstabelle, in der die ganzen Dateien und Verzeichnisse referenziert werden. Der dritte Abschnitt ist ein Duplikat der Dateizuordnungstabelle, die als Sicherheit dient. Eventuell ist es auch möglich, mehrere Kopien der Dateizuordnungstabelle zu erstellen. Danach folgt das Stammverzeichnis und der Rest des Speichers bzw. der 4. Abschnitt wird von den eigentlichen Daten eingenommen. Abbildung 3.2 zeigt die Aufteilung einer Festplatte bei FAT.

Abbildung 3.2.: Aufteilung eines Datenträgers unter Verwendung von FAT.

Addresse	Inhalt	Größe
+00h	Sprungbefehl zur Boot-Routine	3 Bytes
+03h	Herstellername und Versionsnummer	8 Bytes
+0Bh	Bytes pro Sektor	2 Bytes
+0Dh	Sektoren pro Cluster	1 Byte
+0Eh	Anzahl reservierter Sektoren	2 Bytes
+10h	Anzahl File-Allocation-Tables (FAT)	1 Byte
+11h	Anzahl Einträge im Hauptverzeichnis	2 Bytes
+13h	Anzahl Sektoren im Volume	2 Bytes
+15h	Media Deskriptor	1 Byte
+16h	Anzahl Sektoren pro FAT	2 Bytes
+18h	Sektoren pro Spur	2 Bytes
+1Ah	Anzahl der Lese-/Schreibköpfe	2 Bytes
+1Ch	Entfernung des ersten Sektors im Volume vom ersten Sektor auf dem Speichermedium	2 Bytes
+1Eh	Boot-Routine	482 Bytes

Tabelle 3.1.: Der Aufbau des Bootsektors einer mit FAT formatierten Festplatte nach Tischer (1994).

Bootsektor

Da das BIOS beim Starten des Systems als erstes auf den ersten Sektor des zu bootenden Datenträgers zugreift, befindet sich bei FAT an erster Stelle der Bootsektor. Er beinhaltet neben der Boot-Routine auch Daten, welche den zu bootenden Datenträger beschreiben. Es wird dort unter anderm die Größe eines Sektors in Bytes gespeichert, aber auch die Anzahl der FAT-Kopien, Anzahl der Sektoren und vieles mehr. Tabelle 3.1 zeigt den Aufbau des Bootsektors.

Das erste Byte ist Sprungbefehl auf eine spätere Adresse im Bootsektor, wo sich die eigentliche Boot-Routine befindet. Da der Bootsektor doch wesentlich zu kurz ist um

eine komplette Bootroutine dort unterzubringen, verweist der Code im Boodsektor auf irgendeine Datei im Dateisystems, welche die Bootroutine weiterführt. Es wäre auch möglich dem Bootsektor nocht weitere Sektoren anzuschließen, wenn das für den Bootvorgang erforderlich ist. Der Anzahl dieser Sektoren muss im Bootsektor definiert werden. Damit rutscht die FAT und auch das Hauptverzeichnis innerhalb der in 3.2 dargestellten Struktur nach hinten.

File Allocation Table

Die Dateizuordnungstabelle legt fest, auf welchen Sektoren die Dateien des Dateisystems untergebracht sind. Abbildung 3.3 zeigt grob die Structur der FAT-Zuordnungsketten.

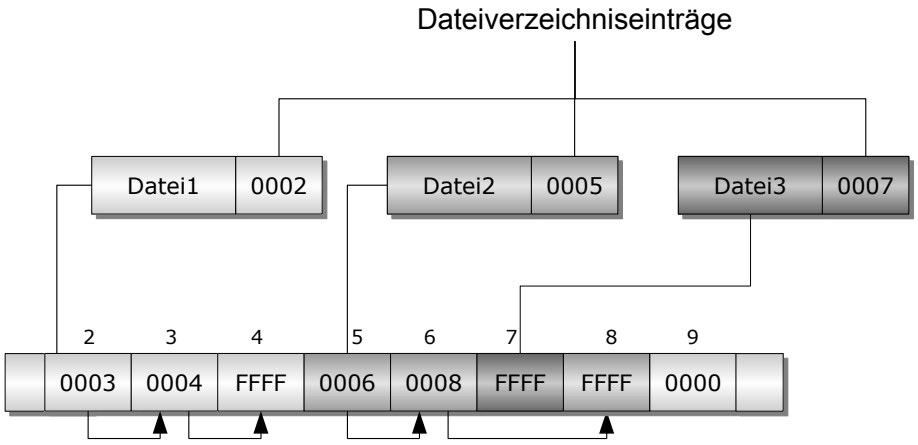


Abbildung 3.3.: grobe Darstellung von FAT-Zuordnungsketten.

Der Datenträger ist hardwaremäßig in Sektoren aufgeteilt. Das Dateisystem FAT faßt eine gewisse Anzahl von Sektoren logisch zu einem Cluster zusammen. Die Größe eines Clusters wird im Bootsektor festgelegt. Die FAT ist nun eine Tabelle, die für jeden Cluster einen Eintrag bereitstellt. Diese Einträge sind je nach FAT Version zwischen 12 und 32 Bit breit. Sie beinhalten im wesentlichen die Information, ob ein Cluster belegt ist oder nicht. Wenn eine Datei mehr Speicher benötigt, als in einem Cluster zur

Cluster-Code 12 Bit FAT	**Bedeutung**
000h	Cluster ist frei
FF0h - FF6h	Reservierter Cluster
FF7h	Cluster fehlerhaft, wird nicht benutzt
FF8h - FFFh	Letzter Cluster einer Datei
xxxh	nächster Cluster einer Datei
Cluster-Code 16 Bit FAT	**Bedeutung**
0000h	Cluster is frei
FFF0h - FFF6h	Reservierter Cluster
FFF7h	Cluster fehlerhaft, wird nicht benutzt
FFF8h - FFFFh	Letzter Cluster einer Datei
xxxh	nächster Cluster einer Datei

Tabelle 3.2.: Kodierung der FAT-Einträge.

Verfügung steh, benötigt, umfaßt der FAT-Eintrag zu dem ersten Cluster auch die Information in welchen Cluster die weiteren Daten gespeichert sind. Diese Kette wird in Abbildung 3.3 veranschaulicht. Neben Adressen eines weiteren Clusters gibt es noch spezielle Codes, die in einem FAT-Eintrag stehen können. Tabelle 3.2 zeigt diese reservierten Codes, aufgeteilt nach FAT12 und FAT16 Dateisystem.

Stammverzeichnis

Das Stammverzeichnis, oder auch Hauptverzeichnis, beinhaltet alle wichtigen Informationen der in ihm befindlichen Dateien und Verzeichnissen. Die Größe des Stammverzeichnisses ist im Bootsektor festgelegt und besteht aus 32 Byte großen Einträgen. In diesen Einträgen werden Informationen wie Name, Größe, Dateityp und auch der Erste Cluster der Dateidaten gespeichert. Tabelle 3.3 zeigt den Aufbau eines solchen 32 Byte großen Directory Eintrags.

Die ersten 11 Bytes des Eintrages sind für Dateinamen und Dateiendung vorgesehen. Ist der Dateiname kürzer als die vorgesehen 8 Bytes oder die Dateiendung kürzer als die dafür vorgesehen 3 Bytes, werden die restlichen Bytes mit Leerzeichen aufgefüllt. Ist der Dateiname leer, dann beschreibt das erste Byte des Feldes für den Dateinamen einen besonderen Eintrag. In Tabelle 3.4 sind die Codes aufgelistet, welche anstelle eines Dateinames in einem Verzeichniseintrags stehen können.

Das Feld Dateiattribut legt die Art der Datei fest. Hier kann zum Beispiel festgelegt

Addresse	Inhalt	Länge
+00h	Dateiname (mit Leerzeichen aufgefüllt)	8 Bytes
+08h	Dateierweiterung (mit Leerzeichen aufgefüllt)	3 Bytes
+0Bh	Dateiattribut	1 Byte
+0Ch	Reserviert	10 Bytes
+16h	Uhrzeit der letzten Verängerung	2 Bytes
+18h	Datem der letzten Veränderung	2 Bytes
+1Ah	Erster Cluster der Datei	2 Bytes
+1Ch	Dateigröße	4 Bytes

Tabelle 3.3.: Aufbau der Verzeichniseinträge bei FAT.

Code	Bedeutung
00h	Letzter Eintrag im Verzeichnis
05h	Das erste Zeichen des Dateinamens hat den ASCII-Code E5h
2Eh	die Datei bezieht sich auf das aktuelle Verzeichnis, folg noch ein 2Eh, bezieht es sich auf das Vater-Verzeichnis
E5h	Datei wurde gelöscht

Tabelle 3.4.: mögliche Status Codes anstelle des Dateinamens.

werden ob eine Datei nur gelesen werden darf, aber nicht beschrieben. Dies bietet keinen wirklichen Schutz, da der Benutzer dieses Attribut jederzeit ändern kann. Es ist mehr dafür da, dass eine Datei nicht versehentlich verändert wird. Weiters ist es möglich eine Datei mithilfe des Attribut Feldes zu verstecken. Die Abbildung 3.4 zeigt die möglichen Attribute einer Datei. Wenn das Bit für Unterverzeichnis gesetzt ist, zeigt dies dem Betriebssystem an, dass es sich hiermit um keine gewöhnliche Datei handelt, sondern um ein Unterverzeichnis, in dem wiederum Dateien sein können.

Ist für eine Datei das Flag *Unterverzeichnis* gesetzt, wird nicht wie bei einer normalen Datei der Cluster mit den Daten referenziert, sondern es wird ein Cluster referenziert, welcher die Verzeichniseinträge des Unterverzeichnisses enthällt. Diese Verzeichniseinträge haben die selbe Struktur wie auch im Hauptverzeichnis.

Datenbereich

Der Datenbereich im FAT beinhaltet die einzelnen referenzierten Cluster. Es können Cluster sein, welche den Inhalt von Dateien repräsentieren, aber es können auch Cluster sein, die weitere Verzeichniseinträge für Unterverzeichnisse enthalten.

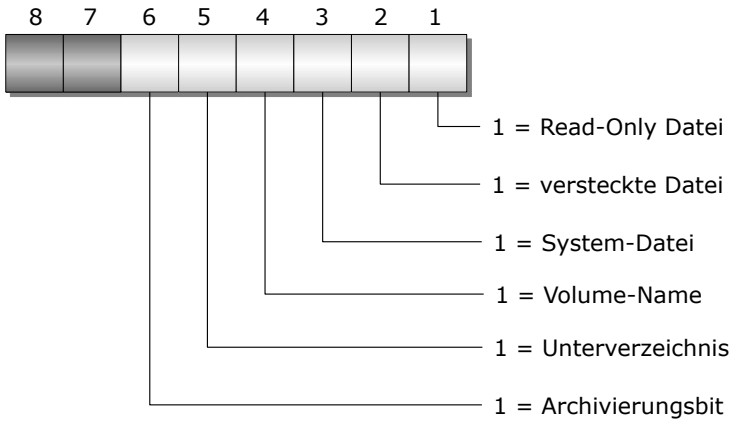


Abbildung 3.4.: mögliche Attribute einer Datei.

3.1.2. NTFS

Das NTFS ist das aktuelle Dateisystem, das in Microsoft Betriebssystemen Verwendung findet. Es bietet die Unterstützung von größeren Datenträgern als es noch bei FAT möglich war. Zudem wurde beim Design die Sicherheit und Mehrbenutzerfähigkeit in den Fordergrund gestellt. Der Zugriff auf eine Datei wird über die sogenannten **Access Control Lists (ACL)** geregelt. Somit ist das NTFS Dateisystem ein sehr sicheres und modernes Dateisystem für PC Systeme. Der detailliert Aufbau und die Funktionsweise von NTFS ist in der NTFS Technical Reference von Microsoft beschrieben. Ebenso findet sich in Russinovich (2005) eine Beschreibung der Funktionsweise von NTFS unter Windows.

Neben den FAT und NTFS Dateisystemen existieren am PC noch einige mehr, wie zum Beispiel das HFS+ der Firma Apple (Bechtel (2006)). Neben den properitären Dateisystemen gibt es auch einige freie, die unter dem Betriebssystem Linux zur Anwendung kommen. Beispiele hierfür sind das ext2 (Extendet 2) Dateisystem oder auch das JFS (Journaled File System) (Elmenreich (2002)) . Auch speziell für Netzwerke gibt es Dateisysteme wie zum Beispiel das NFS (Network File System).

3.2. Dateien in Chipkarten-Betriebsystemen

In diesem Kapitel wird versucht, die wesentlichen Konzepte der Dateiverwaltung von Chipkarten zu beschreiben, angelehnt an die Beschreibung in Rankl (2007).

Neben der ursprünglichen Aufgabe der Identifikation und Authentifizierung dienen Chipkarten auch als Datenspeicher. Ein Beispiel hierfür ist die österreichische **E-card** (ecard (2008)). Bei den ersten Chipkarten war es mehr oder weniger nur möglich auf fixe Adressen zuzugreifen, in welche Daten geschrieben oder gelesen werden konnten. Hier kann man noch nicht wirklich von einer Dateiverwaltung sprechen.
Die meisten heute erhältlichen Chipkarten verfügen nun auch über ein hierarchisch gegliedertes Dateisystem, angeleht an die Dateisysteme, die von PC Betriebssystemen bekannt sind. Jedoch sind die Dateisysteme von Chipkarten bei weitem nicht so kompex wie die von PC Systemen, da bei einem Dateisystem für PC Systeme wesentlich mehr bedacht werden muss (eine Aufstellung von Dateisystemen von Windows Betriebsystemen sind in Russinovich (2005) beschrieben). Bei einem PC System spielt auch die Dateiberechtigung eine wesentliche Rolle, vor allem wenn es sich um ein Mehrbenutzer System handelt. Abgesehen davon weist ein Chipkarten Dateisystem spezifische Merkmale auf. Das wohl auffälligste ist das Fehlen einer Mensch-Maschine-Schnittstelle. Bei einer PC Dateiverwaltung wird Wert darauf gelegt, dass Dateien und auch Verzeichnisse sprechende Namen bekommen können. Anfangs unter MS-DOS durfte eine Datei einen 8 Zeichen langen Namen besitzen, unter Unix 253 Zeichen (anfangs unter V.3 noch 14 Zeichen). Hier sieht man, dass besonderen Wert auf eine gute *menschliche* Lesbarkeit gelegt wurde. Anders bei den Chipkarten. Dateien werden hier konsequent durch hexadezimale Codes adressiert, auch Kommandos werden durch hexadezimale Codes repräsentiert, ähnlich wie es vielleicht ein geübter PC Systemprogrammierer aus den Anfangszeiten des PC's kennt. Hier sieht man, dass die Kommunikation bei Chipkarten ausschließlich von Machine zu Maschine geplant ist.

Da auf den Speicherverbrauch besonders geachtet wird, ist meist keine umfangreiche Speicherverwaltung implementiert. Das Löschen von Dateien, welches bei einem PC Betriebssystem unumgänglich ist, wird bei Chipkarten Betriebssystemen nicht selbsverständlicherweise unterstützt. Selbst bei Systemen, die das Löschen unterstützen ist

es nicht selbstverständlich, dass der durch das Löschen freigegebene Speicherplatz dann auch tatsächlich wieder verwendet werden kann. Das beruht auf der Tatsache, daß Dateien im Regelfall beim Initialisieren der Chipkarten angelegt werden und im laufenden Betrieb lediglich die Inhalte der Dateien verändert werden. Zudem kommt noch dazu, dass die Speicherhardware sich wesentlich von der im PC verwendeten unterscheidet. In einem PC werden hauptsächlich Festplatten (Magnetplatten, siehe Aufbau von Computer-systemen in Tanenbaum (1999)) verwendet, welche nahezu unendlich oft beschrieben werden können. Anders jedoch bei Chipkarten, hier werden die Daten auf EEPROM abgelegt, welche nicht unendlich oft beschrieben werden können. Hersteller (Stand 2006) garantieren in der Regel um die 1.000.000 Schreibvorgänge, ohne dass es zu Datenverlusten kommt. Da die Not bekanntlich erfinderisch macht, gibt es unter anderem besondere Dateiattribute, die erlauben, dass Informationen redundant und auch korrigierbar abgelegt werden.

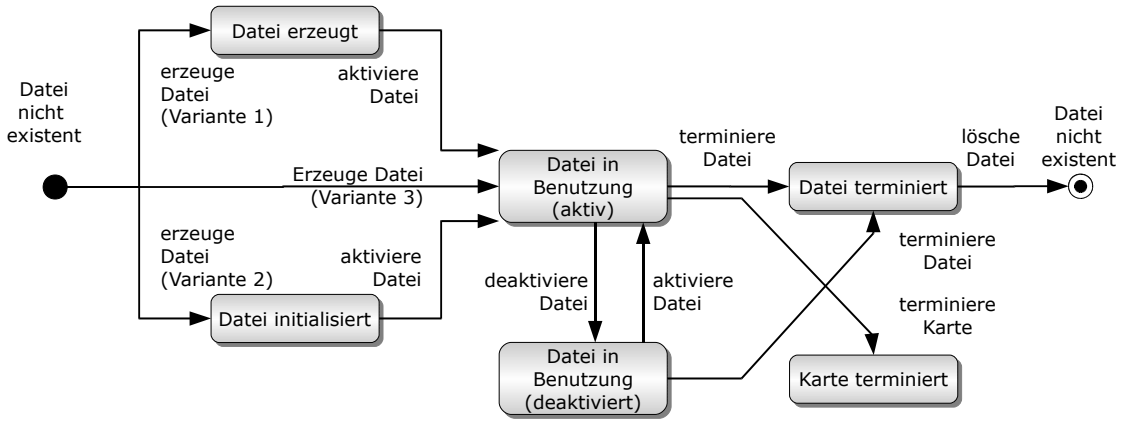


Abbildung 3.5.: Der Lebenszyklus von Dateien in einem Chipkarten Betriebssystem nach ISO/IEC 7816-9 (2004).

3.2.1. Aufbau von Dateien

Aufgrund der objektorientiert aufgebauten moderneren Dateiverwaltungssysteme sind Informationen über eine Datei in ihr selbst gespeichert. Aus diesem Grund muss eine Datei immer zuerst selektiert werden bevor mit ihr gearbeitet werden kann. Darum

sind Dateien in solchen Systemen immer in zwei Teile geteilt. Der sogenannte **Dateiheader** beinhaltet alle notwendigen Verwaltungsdaten zur Datei selbst, wie unter anderem Struktur, Zugriffsbedingungen oder auch den Aufbau der Datei. Ein Zeiger im Dateiheader verweist auf den **Dateibody**, welcher die eingentlichen und auch veränderbaren Nutzdaten speichert.

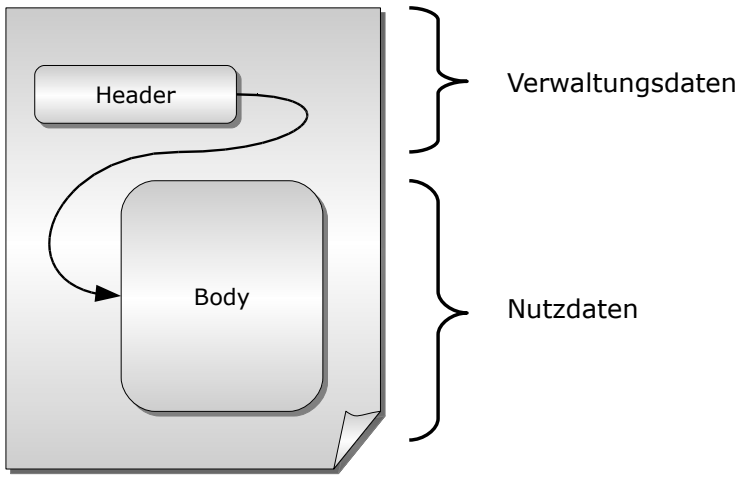


Abbildung 3.6.: interner Aufbau einer Datei bei einem modernen objektorientiert aufgebautem Dateiverwaltungssystem.

Der in Abbildung 3.6 dargestellte interne Aufbau einer Datei eines objektorientierten Dateiverwaltungssystems hat neben der Strukturierung der Daten noch weitere Vorteile.
Da die *Nutzdaten* an einer anderen Stelle als die *Verwaltungsdaten* gespeichert sind, ergibt sich automatisch eine höhere physikalische Sicherheit der Daten. Wie schon zuvor erwähnt, kann ein Datenspeicher, der auf der EEPROM-Technologie basiert nicht unendlich oft gelesen und beschrieben werden. Durch die Trennung von Dateien in *Header* und *Body* können die Verwaltungs- bzw. Nutzdaten auf verschiedenen Speicherseiten auf dem EEPROM abgelegt werden. Dies gewährt weitere Sicherheit für die Daten. In den meisten Fällen wird der Header nicht verändert, da die sich in ihm befindlichen Daten mehr zur Administration (Rechtevergabe usw.) genutzt werden. Angenommen die Verwaltungsdaten (Header) und Nutzdaten (Body) wären sequenziell auf der selben Speicherseite auf dem EEPROM gespeichertn könnten durch einen ge-

zielt erzeugten Schreibfehler die Zugriffsrechte im Header so verändert werden, dass der Zugriff auf geheime Daten ermöglicht wird.

Manche Dateiverwaltungssysteme bieten die Möglichkeit mehrere Header füer eine Datei zu erstellen. Dies bietet die Möglichkeit, auf technisch elegante Weise Daten zwischen mehreren Anwendungen zu teilen (Sharing). Beachtet werden muss dabei aber, dass alle Header die selben Zugriffsbedinungen beinhalten.

3.2.2. Typen von Dateien

Wenn man Dateitypen auf dem PC anschaut, bekommt man den Eindruck, es gäbe unendlich viele davon. Dies ist nicht ohne Grund so, denn mehr oder weniger jede Anwendung speichert ihre spezifischen Daten in Dateien unter Verwendung von properitären Dateiformaten. Aber nicht ausschließlich, denn es gibt auch gewisse Standardformate die sich auf dem Markt durchgesetzt haben. Das sind unter anderem Formate für Textverarbeitungssoftware (DOC), Tabellenkalkulation (XLS), ja sogar für die Erstellung von Webseiten (HTML). Auch die Dateisysteme sind meist von jedem Hersteller eines Betriebssystems selbst entwickelt.

Anders bei Chipkarten Betriebssystemen. Der Aufbau vom Dateisystem ist in der ISO/IEC 7816-4 (2005) genau festgelegt. Dieser Aufbau ähnelt stark dem Aufbau des Dateisystems von Unix (siehe 3.1) oder auch dem von MS-DOS. Dennoch gibt es wesentliche Unterschiede. Der größte Unterschied besteht darin, dass es bei Chipkarten Dateisystemen keine anwendungsspeziefischen Dateien gibt, wie zum Beispiel eine Datei für eine Tabellenkalkulation.

Wie in Abbildung 3.7 dargestellt, sieht die Norm ISO/IEC 7816-4 (2005) zwei Kategorien von Dateien vor. Der Typ der ,,Verzeichnisdateien‘‘, welche als **Dedicated Files (DF)** bezeichnet werden, und der Typ der ,,Datendateien‘‘, mit der Bezeichnung **Elementary Files (EF)**. Die DFs können als eine Art Ordner angesehen werden, welche wiederum DFs beinhalten können und auch EFs, welche durch die Tatsache, dass sie sich im selben DF befinden, logisch zusammengehören. Die EFs lassen sich in zwei

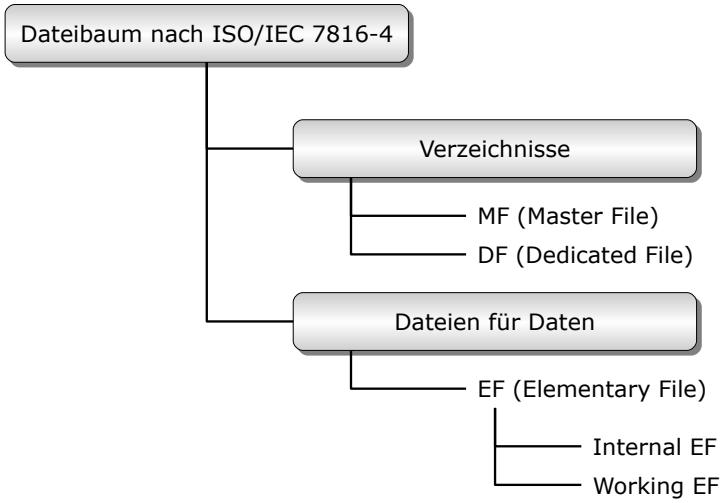


Abbildung 3.7.: Dateitypenklassifizierung für Chipkarten Betriebssystemen nac ISO/IEC 7816-4 (2005).

Kategorien einteilen. Auf der einen Seite in Dateien für die äußere Welt (working EF) und solche, die für das Betriebssystem reserviert sind (internal EF).

In Abbildung 3.8 erkennt man recht leicht, dass die Struktur des Dateibaums dem von PC Betriebssystemen, wie MS-DOS oder Unix, sehr ähnlich sieht.

MF

Nach dem Reset der Chipkarte wird automatisch das Root-Verzeichnis des Dateisystems selektiert. Somit muss das Root-Verzeichnis auf jeder Fall vorhanden sein. Bei einer Chipkarte wird das Root-Verzeichnis als **Master File (MF)** bezeichnet. In diesem Verzeichnis befinden sich alle anderen Verzeichnisse und auch alle anderen Dateien.

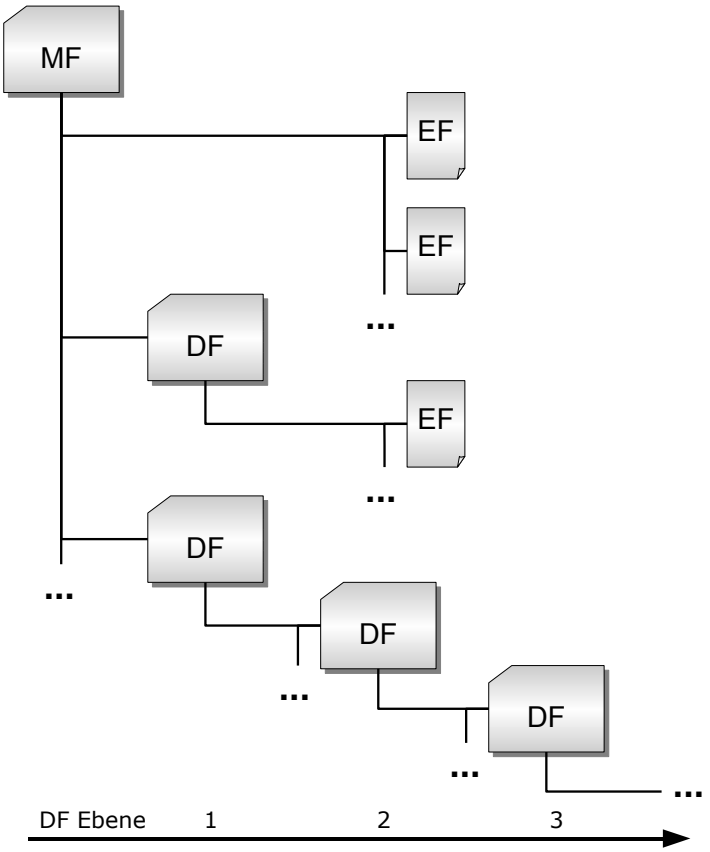


Abbildung 3.8.: In einem Dateibaum dargestellten Dateitypen.

DF

Unter diesem Root-Verzeichnis können noch weitere Verzeichnisse erstellt werden. Diese werden laut ISO/IEC 7816-4 (2005) als **Dedicated Files** bezeichnet und als **DF** abgekürzt. Oft wird die Abkürzung DF auch als *Directory File* interpretiert, was zwar den Sinn der Files wiedergibt, aber laut ISO/IEC 7816-4 (2005) nicht vorgesehen ist. In jedem dieser DF können auch wieder andere Verzeichnisse und auch Dateien vorhanden sein. Somit lassen sich theoretisch beliebig viele Verzeichnisse in ein Verzeichnis packen. Ganz beliebig ist die Verzeichnistiefe jedoch nicht, da auf einer Chipkarte der Speicherplatz sehr beschränkt ist. Somit wird in der Praxis selten eine Verschachtelungstiefe von mehr als zwei Ebenen unter dem Root-Verzeichnis (MF) erstellt.

Im Mobilfunkbereich (GSM und UMTS) wird die sogenannte **Universal Integrated Circuit Card (UICC)** verwendet. In der für die UICC festgelegten technischen Spezifikation ETSI TS 102.221 (2004) findet sich eine Sonderform der DFs. Es wird eine **Application Dedicated File** beschrieben, welche eine Art Root-Verzeichnis speziell für Anwendungen darstellt und nicht dem MF untergeordnet ist.

EF

Wir haben nun zwei Typen von Verzeichnissen kennengelern, mit denen sich das Dateisystem gliedern läßt, jedoch noch keine tatsächlichen Dateien zum Daten speichern. Diese Aufgabe übernehmen die so genannten **Elementary Files (EF)**. Wie schon erwähnt, können EFs direkt einem MF untergeordnet sein. Sie können aber auch in ein DF geschrieben werden. Wie schon in 3.6 dargestellt, haben EFs eine normierte interne Dateistruktur und können im Gegensatz zu ihren Pendanten am PC keine von einer Anwendung vorgegebene Struktur erhalten. Die Gruppe der EFs kann wiederum in zwei Typen aufgeteilt werden:

Working EF

Nutzdaten, also soche Daten, die von Anwendungen verwendet werden und von einem Terminal gelesen oder geschrieben werden müssen, sind in **Working EFs** abgelegt. Sie sind also, aus der Sicht der Chipkarte, nach außen sichtbar. Diese Dateien werden ausschließlich von Anwendungen und von einem Terminal benutzt.

Internal EF

Daten, die nach außen nicht sichtbar sein dürfen, wie Systemdateienn in denen geheime Schlüssel, Programmcodes oder ähnliches abgelegt sind, werden in **Internal EFs** gespeichert. Diese Dateien lassen sich von Anwendungen nicht selektieren, wofür

das Betriebssystem sorgt. Nach der ISO/IEC 7816-4 (2005) Norm gibt es zwei verschiedene Möglichkeiten, interne Systemdateien in die Dateiverwaltung zu integrieren. Die Systemdateien bekommen einen ganz normalen Dateinamen und sind somit auch selektierbar. Diese Variante wird auch von MS-DOS praktiziert. Die zweite Möglichkeit wäre, die Dateien in dem dazu passenden Anwendungs DF zu integrieren. Da die Datei versteckt ist, kann sie dann auch nicht selektiert werden und wird transparent vom Betriebssystem verwaltet. Diese Variante ist dem Konzept von Ressource-Dateien (DS_Store) unter MacOS (Bechtel (2006)) ähnlich, in denen unter anderem Informationen wie Fenstergröß beim Anzeigen eines Ordners oder ähnliches gespeichert werden können. In MacOS X sind diese Dateien für den Benutzer versteckt, können jedoch von Anwendungen aber auch von technisch versierten Benutzern gelesen oder auch verändert werden.

Üblicherweise werden EFs einer Anwendung direkt in das DF der jeweiligen Anwendung gespeichert. Da auf einer Chipkarte oftmals nur eine einzige Anwendung untergebracht ist, und ein MF eine Spezialform der DFs ist, werden oft die Daten auch direkt in das MF gespeichert. Wenn mehrere Anwendungen auf einer Chipkarte untergebracht sind, macht eine Aufteilung in mehrere DFs durchaus Sinn. Auch innerhalb eines DFs einer Anwendung kann es Sinn machen, eine Unterteilung einzuführen. Wenn, zum Beispiel die Anwendung mehrere Sprachen unterstützt, könnte man die Daten für die einzelnen Sprachen in separate DFs speichern.

Abbildung 3.9 zeigt eine typische Dateiorganisation, wie sie bei Chipkarten verwendet wird, auf denen nur eine Anwendung zum Einsatz kommt.

In Abbildung 3.10 ist eine Dateiorganisation mit zwei oder mehreren Anwendungen dargestellt, wobei eine Anwendung auch wieder zwei oder mehrere DFs verwendet, um beispielsweise eine Unterstützung für mehrere Sprachen zu verwalten.

3.2.3. Dateinamen

Um nun auf die zuvor beschriebenen Dateitypen zugreifen zu können, benötigen wir eine Bezeichnung für die jeweilige Datei. In frühren Betriebssystemen für Chipkarten wurden Dateien noch direkt über eine physikalische Adresse angesprochen. Sobald

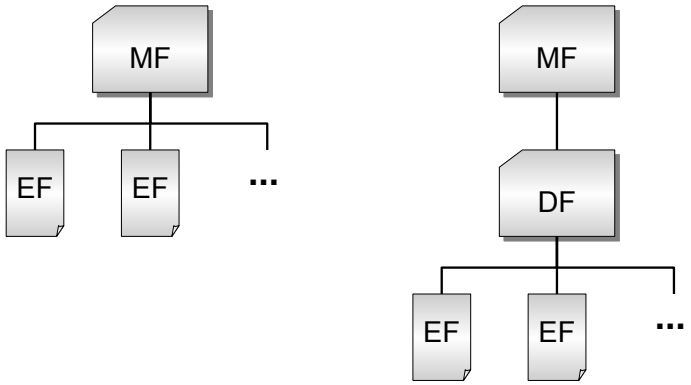


Abbildung 3.9.: typische Dateiorganisation für Chipkarten mit nur einer Anwendung.

eine Anwendung ein wenig komplexer wird, ist diese Art des Zugriffs sehr mühsam aber auch fehleranfällig. Die modernere Variante ist die Verwendung von logischen Adressen für eine Datei. Diese Variante stellt mit Sicherheit die Zukunft dar und wird über kurz oder lang die direkte Adressierung komplett verdrängt haben, eine Ausnahme mögen vielleich Speicherkarten sein.

File Identifier (FID)

Die Norm ISO/IEC 7816-4 (2005) beschreibt eine Variante, wie Dateien benannt werden können. Hierfür besitzen alle Dateien, inklusive des MF, einen 2 Byte langen **File Identifier (FID)**, der als Name der Datei dient. Unter seiner Verwendung kann die Datei ausgewählt werden. Das Root-Verzeichnis, also der MF, besitzt den Wert '3F00'. Dieser Wert darf ausschließlich vom MF verwendet werden. Es gibt eine Reihe weiterer vordefinierter (reservierter) Werte, welche in Tabelle 3.5 aufgelistet sind.

Bei der GSM Anwendung ist es klar ersichtlich, dass ein FID nicht frei gewählt werden darf. Laut der ETSI GSM 11.11 (1995) Spezifikation bestimmt das höherwertige Byte das DF, unter welchem die Datei abgespeichert wird.

Eine wichtige Datei stellt das EF_{DIR} mit der FID '2F00' dar. Sie beschreibt die auf der

FID	Verwendung
'2F00'	Diese FID wir für die Datei EF_{DIR} verwendet, in welcher die AIDs einer Anwendung gespeicher werden.
'2F01'	Diese FID ist für die Datei EF_{ATR} reserviert.
'3F00'	Diese FID ist für das Root-Verzeichnis (MF) reserviert.
'3FFF"	Diese FID ist für die Dateiselektion durch Pfadangabe reserviert.
'FFFF"	Diese FID ist für die zukünftige Benutzung reserviert.

Tabelle 3.5.: Die wichtigsten reservierten FID-Werte nach ISO/IEC7816-4.

EF_{DIR}	Verzeichnis-EF (dir-directory)	
Beschreibung	Diese Datei enthält Informationen über die auf einer Chipkarte befindlichen Anwendungen.	
Datei	FID = '2F00'; Struktur: linear fixed, Dateigröße: n Bytes Zugriffe: READ: immer; UPDATE: je nach Anwendung, im Allgemeinen jedoch nur Administrator	
Codierung eines Records	Byte 1:	Template für Anwendung '61'
	Byte 2:	Länge des Template für Anwendung (3 ... 127)
	Byte 3:	Kennzeichen des AID '4F"
	Byte 4:	Länge des AID (1 ... 16)
	Byte 5 ... n:	AID
	Byte n + 1:	Kennzeichen der Anwendungsbezeichnung '50'
	Byte n + 2:	Länge der Anwendungsbezeichnung
	Byte n + 3 ... m:	Anwendungsbezeichnung in ASCII (0 ... 16)
Beispiel	'61 0F 4F 05 D2 76 00 00 60 50 05 52 61 6E 6B 6C'	
	'61'	⇒ Template für Anwendung
	'0F"	⇒ Länge des Templates ⇒ Länge = 15 Bytes
	'4F"	⇒ Kennzeichen des AID
	'05'	⇒ Länge des AID ⇒ Länge = 5 Bytes
	'D2 76 00 00 60'	⇒ AID
	'4F"	⇒ Kennzeichen der Anwendungsbezeichnung
	'05'	⇒ Länge der Anwendungsbezeichnung
		⇒ Länge = 5 Byte
	'52 61 6E 6B 6C'	⇒ Anwendungsbezeichnung = „Rankl“

Tabelle 3.6.: Beispiel von Rankl (2007) für einen typischen Aufbau eines EF_{DIR}.

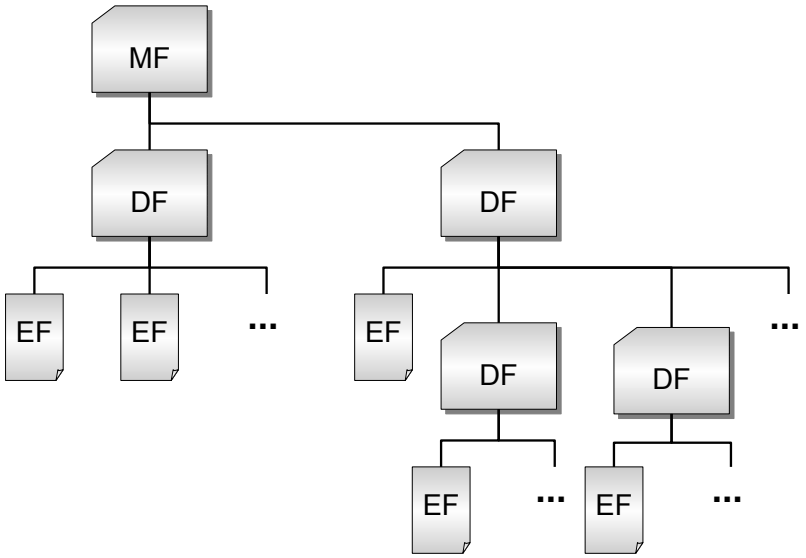


Abbildung 3.10.: typische Dateiorganisation für Chipkarten mit zwei oder mehreren Anwendungen.

Chipkarte befindlichen Anwendungen. Damit lässt sich erreichen, dass das Terminal somit in standardisierter Form die Information über die auf der Chipkarte befindlichen Anwendungen erhält. Tabelle 3.6 zeigt ein Beispiel von Rankl (2007) wie ein EF_{DIR} ausschauen könnte.

FIDs müssen, wie auch die Dateinamen bei PC Betriebssystemen, eindeutig gewählt werden. Die Eindeutigkeit muss nicht auf dem gesamten Dateisystem gewährleistet sein, es reicht, wenn die Eindeutigkeit im DF, der sich befindlichen EF, gewährleistet ist. Genau so wenig dürfen ein EF und ein DF dieselbe FID besitzen. Eine Besonderheit gegenüber den meisten Dateisystemen von PCs ist die Tatsache, dass untergeordnete DFs nicht den selben FID besitzen dürfen wie das übergeordnete FD.

Eine weitere Möglichkeit zur Selektion von EFs ist der **Short File Identifier (SFI)**. Er erlaubt es unmittelbar in einem Kommando ein EF zu adressieren. Ein EF muß keine SFI besitzen, sie ist optional. Sie ist nur 5 Bit lang und kann Werte zwischen 1 und 30 annehmen. Der Wert '0' ist reserviert und adressiert das akuell selektierte EF.

DF Name

DFs beinhalten EFs oder auch weitere DFs. Sie sind somit vergleichbar mit Verzeichnissen und dienen zur Gliederung der Daten. Da der Speicherplatz auf Chipkarten und auch deren Anwendungen immer größer werden, könnte der Adressraum des FIDs von 2 Bytes in Zukunft bald zu klein werden. Aus diesem Grund sieht die ISO/IEC 7816-4 (2005) einen zusätzlichen *DF Name* vor, der eine Länge zwischen 1 und 16 Byte einnehmen darf. Um noch weiter einem doppelten Namen vorzubeugen wird der DF Name in der Regel nur in Verbindung mit dem **Application Identifier (AID)** (nach ISO/IEC 7816-5 (2004)) verwendet. Der AID wiederum setzt sich aus der PIX und dem RID zusammen und kann somit eine Länge von 5 bis 16 Byte haben, da die PIX optional ist und somit zwischen 0 und 11 Byte lang sein darf. Abbildung 3.11 veranschaulicht diesen Aufbau.

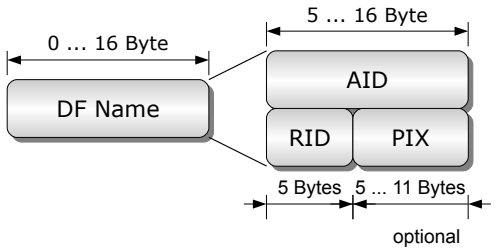


Abbildung 3.11.: Der Aufbau des DF Namens in Verbindung mit dem RID und der PIX.

Der RID kann nicht frei vergeben werden, er wird von einer nationalen oder internationalen Registrierungsbehörde vergeben. Jeder Anwendungsanbieter bekommt eine eigene Nummer. Diese wird zusammen mit einem Ländercode und einem Code für die Anwendungskategorie als RID eingetragen. Mit dieser RID kann weltweit ein Anwendungsanbieter und die Anwendung identifiziert werden. RIDs sind vertraulich und werden nicht der Öffentlichkeit zugänglich gemacht.
Wenn der Anbieter es wünscht, kann er der RID noch eine bis zu 11 Byte lange PIX

D1	D2 ... D4	D5 ... D10	Bedeutung
X	...	...	Kategorie der Registrierung 'A' - Internationale Registrierung 'D' - Nationale Registrierung
...	X		Ländercode, Codierung nach ISO 3166
...	...	X	Nummer des Anwendungsherstellers, wird von der Registrierungsinstanz vergeben

Tabelle 3.7.: Codierung des 5 Byte (= 10 Digits) RID (Registered Identifier).

anhängen um zum Beispiel Serien oder Versionsnummer der Anwendung anzugeben. Die Tabelle 3.7 zeigt die Codierung des RID.

3.2.4. Dateistrukturen von EFs

Dateien haben in PC Betriebssystemen keine vordefinierte Struktur. Der Aufbau einer Datei ist jedem Anwendungsentwickler frei gestellt. Anders bei Chipkarten Systemen. Hier hat jedes EF eine von mehreren vordefinierten Strukturen. Ziel dabei ist es, den Zugriff auf Dateien effizienter zu machen, wenn fixe Strukturen vordefniert sind. Ein Nachteil, der dadurch entsteht, ist der zusätzliche Programmcode, der notwendig wird. Abbildung 3.12 zeigt die Klassifizierung der Dateistrukturen, die bei Chipkarten Betriebssystemen vorkommen.

transparent

Dateien, welche nach der als *transparent* (siehe 3.13) bezeichneten Datenstruktur aufgebaut sind, können mit Dateien von PC Betriebssystemen verglichen werden. Diese Datenstruktur wird oft auch als binäre Struktur bezeichnet. Wie auch bei PC Systemen lassen sich diese Dateien byteweise oder auch blockweise lesen. Dafür sind die Kommandos 'READ BINARY', 'WRITE BINARY' und 'UPDATE BINARY' vorgesehen. Leere Dateien, also Dateien mit einer Länge von 0 Bytes, sind, anders als bei manchen PC Betriebssystemen, nicht vorgesehen. Jede Datei hat somit eine Mindestlänge von 1 Byte. Die Maximallänge von Dateien ergibt sich durch die maximale Anzahl der schreibbaren Dateien im *Extended Format* von 16 Bit (65 536) und dem maximalen Offset von 15

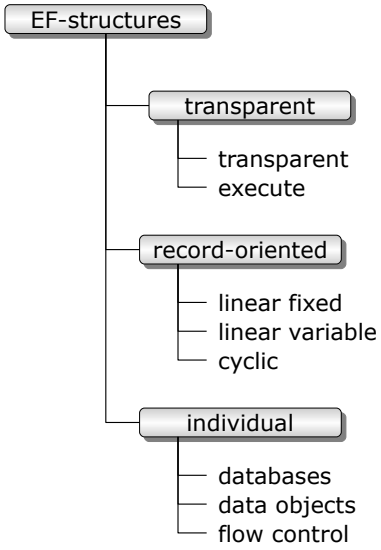


Abbildung 3.12.: Dateistrukturen für EFs bei Chipkarten Betriebssystemen.

Bit (32767). Daraus resultiert eine maximale Dateigröße von 98303 Bytes.

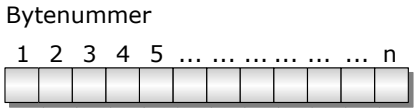


Abbildung 3.13.: Aufbau einer Datei mit der Dateistruktur 'transparent'.

Der Zugriff auf eine Datei mit der Struktur *transparent* erfolgt durch die Angabe eines Offsets und der Anzahl der zu lesenden Bytes. Genau auf diese Art werden auch Dateien in PC Betriebssystemen gelesen und geschrieben. Abbildung 3.14 zeigt den Zugriff auf ein EF mit der Länge von 10 Bytes, von wo 5 Bytes ab dem 2. Byte gelesen werden.

Dieser Aufbau einer Datei ist sehr flexibel und es lassen sich damit alle erdenklichen Dateistrukturen nachbauen. Dies ist jedoch mit einem Mehraufwand am Terminal ver-

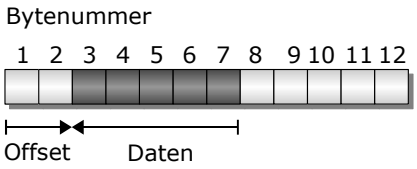


Abbildung 3.14.: Beispiel für das Lesen von 5 Bytes ab dem Offset 2.

bunden und führt zu Geschwindigkeitseinbußen und einer höheren Komplexität der Anwendung.

linear fixed

Hier wird ein sogenannter *record* definiert, welcher eine Länge von 1 Byte bis maximal 254 Byte besitzen kann. Es kann jetzt nur immer auf die einzelnen *records* zugegriffen werden, jedoch nicht direkt auf deren Inhalt. Dieser Aufbau ist ähnlich dem eines Datensatzes einer Datenbank. Ein Beispiel hierfür ist eine Kundendatenbank. Ein *record* beschreibt den Datensatz eines einzelnen Kunden. Alle Datensätze haben eine einheitliche Länge, wobei natürlich nicht genutzter Platz verschenkt wird (z.B.: Nicht jeder Kundennname hat die selbe Länge). In Abbildung 3.15 ist die Struktur von *linear fixed* dargestellt.

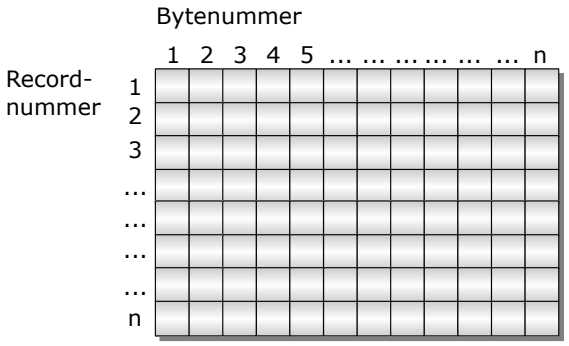


Abbildung 3.15.: Der Aufbau einer Datei mit der Datenstruktur 'linear fixed'.

linear variable

Da der Speicherplatz auf einer Chipkarte ein wertvolles Gut darstellt, und Speicher nur sehr ungern verschenkt wird, wie es bei *records* bei der Dateistruktur *linear fixed* vorkommen kann, gibt es eine weitere Möglichkeit Dateien zu strukturiern. Bei 'linear variable' darf jeder *record* eine eigene Länge besitzen. Somit kann vermieden werden, dass bei unterschiedlich langen Datensätzen wertvoller Speicherplatz verschenkt wird. Ganz ohne Nachteil ist diese Dateistruktur nicht, denn die Verwaltung dieser Datenstruktur bedeutet einen größeren Verwaltungsaufwand, was sich natürlich auch in der Codegröße niederschlägt. Abgesehen von der variablen Länge eines *records* unterscheidet sich 'linear variable' nicht von 'linear fixed'. Bei beiden Datenstrukturen wird mit den Kommandos 'READ RECORD', 'WRITE RECORD' und 'UPDATE RECORD' zugegriffen. In Abbildung 3.16 ist die Dateistruktur 'linear variable' dargestellt.

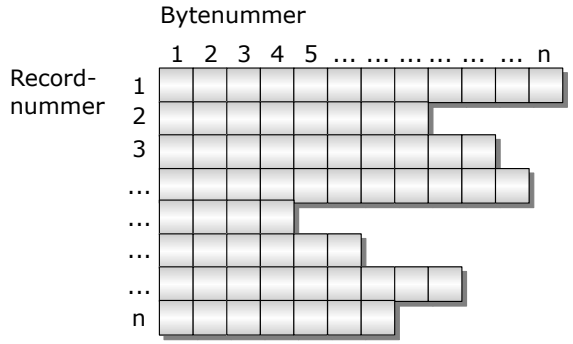


Abbildung 3.16.: Der Aufbau einer Datei mit der Datenstruktur 'linear variable.

cyclic

Die Dateistruktur *cyclic* basiert auf der Struktur *linear fixed*. Im Gegensatz dazu wird bei *cyclic* nicht direkt auf einen *recod* zugegriffen. Stattdessen gibt es einen Zeiger, der immer auf den aktuellen *record* zeigt. Der aktuellste (neueste) Datensatz hat immer die Nummer 1. Je älter ein Datensatz, umso höher seine Nummer. Der als erstes geschriebene Datensatz hat die Nummer n. Der Zeiger auf den aktuellen Datensatz kann zum

nächsten (*next*), zum vorherigen (*previous*), zum ersten (*first*) und zum letzten (*last*) Datensatz navigiert werden. Wird der letzte Datensatz erreicht, wird vom Betriebssystem der Datensatzzeiger nach dem Lesen oder Schreiben automatisch wieder auf den ersten Datensatz gesetzt. Dadurch werden bei endlosem Schreiben die Datensätze immer wieder überschrieben. Dies Art von Datenstruktur eignet sich sehr gut für 'Logging'. Abbildung 3.17 zeigt schematisch den Aufbau der Dateistruktur 'cyclic'.

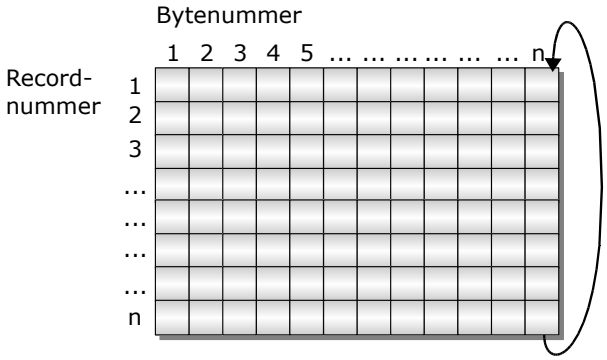


Abbildung 3.17.: Der Aufbau einer Datei mit der Datenstruktur 'cyclic'.

execute

Die Datenstruktur *execute* ist an und für sich keine eigenständige Dateistruktur, denn eine Datei, die als *execute* angelegt ist, hat die selbe interne Struktur wie *transparent*. Sie ist nicht für die Speicherung von Daten vorgesehen, sondern zum Ablegen von Programmcode. Zugegriffen werden kann auf diese Dateistruktur anolog wie auf eine Datei mit der Struktur *transparent*. Es ist aber Vorsicht geboten, denn dadurch wird die Möglichkeit geschaffen fremden Programmcode einzuschleusen oder trojanische Pferde auf der Karte abzulegen.

weitere Dateistrukturen

Die ISO/IEC 7816-7 (1999) beschreibt mit **SCQL** eine spezielle Variante von SQL für Chipkarten. Weiters können mit den Kommandos 'GET DATA' und 'PUT DATA' nach ISO/IEC 7816-4 (2005) **TLV**-codierte Datenobjekte auf Chipkarten abgelegt werden. Auch für die Ablaufsteuerung wird eine besondere Dateistruktur verwendet, diese ist aber nicht genormt und jedes Betriebssystem verwendet sein eigenes Format.

3.2.5. Selektion von Dateien

Anders als bei PC Betriebssystemen gibt es bei Chipkarten-Betriebssystemen immer nur eine selektierte Datei. Bei PC Betriebsystemen wie Unix oder MS-DOS kann jede Anwendung theoretisch beliebig viele Dateien geöffnet haben. Anders bei Chipkarten. Hier wird, wie schon erwähnt beim Reset automatisch das MF selektiert. Wenn nun ein DF oder EF, welches sich direkt im MF befindet, selektiert wird, ist die Selektion des MF automatisch aufgehoben. Des Weiteren können von einem DF aus nicht beliebig Dateien geöffnet werden. Es können nur solche EFs geöffnet werden, welche direkt in dem zuvor selektierten DF liegen. Auch DFs können nicht beliebig gewählt werden, hier muss sich ein zu selektierendes DF entweder im zuvor selektierten DF befinden, oder auf der selben Ebene wie das zuvor selektierte DF. Dieser Umstand beruht auf der Tatsache, dass ein FID immer nur auf einer Ebene eindeutig definiert ist. PC Betriebssysteme lösen den Zugriff auf Dateien auf eine andere Art. Dort wird eine Datei nicht nur durch den Dateinamen adressiert sondern auch durch einen Pfad, welcher die Position der Datei innerhalb des Verzeichnisbaums eindeutig beschreibt. 3.18 zeigt exemplarisch gültige Zugriffe auf Dateien, jeweils von einem Ursprungsverzeichnis aus. In Abbildung 3.19 sind einige Zugriffe dargestellt, wie sie von Chipkarten-Betriebsystemen nicht erlaubt sind.

Selektion von Verzeichnissen

DFs können entweder über ihren FID ausgewählt werden oder über den DF Name des Verzeichnisses. Das MF, ein Sezialfall eines DFs, kann entweder durch die Angabe einer speziellen Option des Selektionskommandos ausgewählt werden oder aber auch

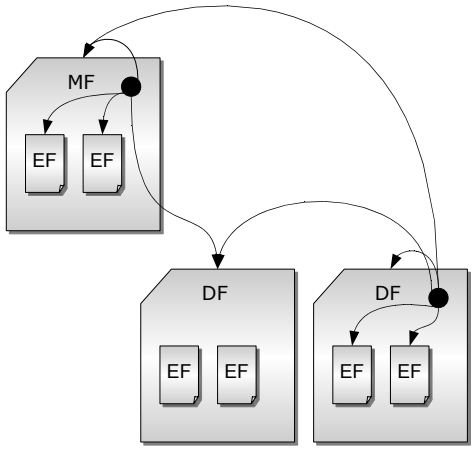


Abbildung 3.18.: Beispiele für gültige Zugriffe unter Verwendung des FID

durch den nur einmal vorkommenden FID '3F00', welcher eindeutig das MF kennzeichnet.

Explizite Selektion von EFs

Ein EF kann durch die Angabe des FID als Parameter des Selektionskommandos ausgewählt werden. Das Kommando *'SELECT FILE'* erlaubt als Parameter den 2 Byte langen FID, welcher das EF kennzeichnet. Nach dieser Selektion kann auf das EF weiter mit anderen Kommandos zugegriffen werden.

Implizite Selektion von EFs

Eine weitere Variante der Selektion von EFs ist die implizite Selektion. Hierbei wird eine Datei nicht direkt durch das Kommando *'SELECT FILE'* ausgewählt, sonder es wird ein anderes Kommando direkt aufgerufen, welches als Parameter einen Short Identifier entgegennimmt. Dies ist aber nicht mit jedem Kommando möglich. Als weitere Einschränkung gilt, dass nur EFs durch solche Kommandos (z.B.: READ /UPDATE BINARY

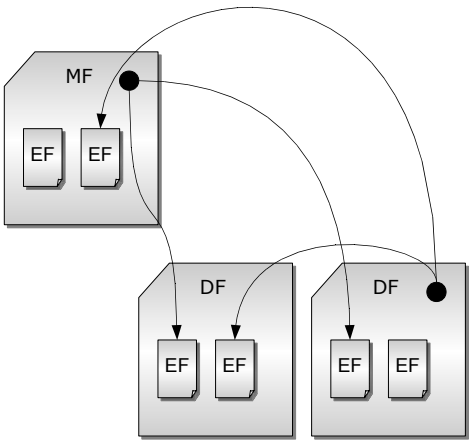


Abbildung 3.19.: Beispiele für ungültige Zugriffe unter Verwendung des FID

/RECORD) implizit selektiert werden können, die sich auch im aktuell selektierten MF oder DF befinden. Ein großer Vorteil der impliziten Selektion ist die Ausführungsgeschwindigkeit. Durch die Angabe eines Short Identifiers bei einem Kommando entfällt das zusätzliche Kommando der Selektion, somit fällt ein Kommando weg, was die Ausführungsgeschwindigkeit erhöht.

Selektion durch die Angabe eines Dateipfades

Ähnlich wie schon bei PC-Betriebssystemen erwähnt, existierten nach ISO/IEC 7816-4 (2005) zwei weitere Zugriffsmethoden, die es erlauben unter Angabe eines Dateipfades eine Datei zu selektieren. Dies führt zu einer höheren Ausführungsgeschwindigkeit, da wieder Kommandos gespart werden können. Es werden hier zwei Arten des Zugriffs mit Hilfe einer Pfadangabe unterschieden. Einerseits durch die Angabe eines *absoluten Pfades*. Hier wird ein Pfad relativ zum MF angegeben. Auf der anderen Seite läßt sich die Adresse auch durch einen *relativen Pfad* angeben. Mit 'relativ' ist die Angabe des Pfades aus der Sicht des aktuell selektierten DFs gemeint.

Information	Beispiel
Name der Datei	'0001'
Type der Datei	EF
Größe der Datei	3 Records mit je 10 Bytes
Zugriffsbedingungen	READ erst nach PIN Eingabe
Dateiattribute	WORM
Position im Dateibaum	direkt im MF

Tabelle 3.8.: Daten die ein Dateideskriptor mindestens enthalten muss.

3.3. Verwaltung von Dateien

Da eine Chipkarte ja mobil ist und immer nur Strom bekommt, wenn sie in einem Terminal steckt, muss es einen Speicher auf der Chipkarte geben, welcher Daten auch ohne Stromversorgung längere Zeit speichern kann. Aus diesem Grund besitzen Chipkarten ein EEPROM. Hier werden alle Daten gespeichert, die über eine Sitzung hinaus behalten werden sollen. Bei früheren Betriebssystemen war der Speicher einer Chipkarte von außen direkt adressierbar. Dies ist allerdings aus Sicht der Sicherheit kritisch. Heutige Betriebssysteme implementieren ein eigenes Dateiverwaltungssystem um die Daten vor unbefugtem Zugriff zu schützen. Zudem ist es durch eine Dateiverwaltung einfacher größere Datenmengen zu verwalten. Eine Dateiverwaltung ist bei PC Betriebssystemen schon lange üblich. In Russinovich (2005) sind die Dateiverwaltungssysteme von Microsoft Betriebssystemen genau beschrieben.

3.3.1. Dateideskriptor

Wie schon in Kapitel 3.2.1 beschrieben bestehen Dateien, so wie sie in der Chipkartentechnik verwendet werden, aus einem *Header* und einem *Body*. Alle Daten, die für die Dateiverwaltung relevant sind, werden im Header der Datei untergebracht. Der *Dateiheader* wird auch als **Dateideskriptor** bezeichnet.

Die Tabelle 3.8 zeigt den Minimalinhalt eines Dateideskriptors. Der Name der Datei ist frei wählbar, jedoch gibt es reservierte Namen, die nicht verwendet werden dürfen.

Tabelle 3.5 zeigt ein paar reservierte Dateinamen, die nicht verwendet werden dürfen. Bei EFs oder dem MF ist der Name der Datei 2 Byte lang. Bei DFs kommt zu diesem Namen noch ein AID (application identifier) hinzu. Aus diesem Grund muß der Typ einer Datei auch angegeben sein. Der Typ kann EF, DF oder MF sein, wobei MF nur einmal im Dateisystem vorkommen darf. Ist der Dateityp EF, muß noch ein Datenelement im Dateideskriptor existieren, welches die interne Struktur der Datei beschreibt. Hier sind Strukturen wie z.B.: *transparent*, *linear fixed*, *linear variable*, *cyclic* oder auch *executeable* möglich. Wiederum abhängig von der internen Struktur einer Datei und auch den Daten ergibt sich eine Länge der Datei. Je nach Struktur wird entweder die Gesamtlänge in Bytes (z.b.: bei transparent) gespeichert oder die Anzahl der Records (z.B.: linear fixed). Neben den grundlegenden Eigenschaften einer Datei werden auch Informationen zu Zugriffsbedingungen der Datei gespeichert. Bei Chipkarten ist das der Zustand, in dem sich das System befinden muss, damit auf die Datei zugegriffen werden darf. Zum Beispiel ist es bei einer Bankkarte sinnvoll, dass sich die Karte in einem Zustand *nach* der erfolgreichen PIN-Eingabe befindet, bevor auf Dateien zugegriffen werden darf. Ein Chipkarten-Betriebssystem erlaubt die Zugriffsbedingung auf eine Datei für jedes Kommando separat festzulegen. Je nach Chipkartenbetriebsystem können noch weitere Attribute festgelegt werden (z.B.: WORM). Damit das Dateisystem auch weiß wo sich die Datei im Dateisystem zu befinden hat, wird noch ein Zeiger im Dateideskriptor definiert, der die Lage der Datei im Dateisystem beschreibt.

3.3.2. Dateiverwaltung mit Zeiger

Um den Verwaltungsaufwand des Dateiverwaltungssystems auf ein Minimum zu begrenzen, verwenden einfachere Chipkarten-Betriebssysteme, je nach Dateityp, Dateiheader von fester Länge. Dadurch ist ein Dateisystem, welches eine feste Länge der Header verwendet, recht unflexibel in Hinsicht auf Erweiterungen. Ebenfalls die Festlegung von Zugriffsbedingungen stellt sich als unvorteilhaft heraus. Da für jedes Kommando die Zugriffsbedingung separat festgelegt wird, aber auch nicht jede Datei jedes Kommando unterstützen muss, wird hier viel Speicher verschwendet.
Um diese Nachteile zu umgehen, verwenden Chipkartenbetriebssysteme oft Dateiheader mit variabler Länge. Damit wird kein Speicherplatz verschenkt, wenn eine Datei nicht alle Kommandos unterstützt und somit auch keine Zugriffsbedingungen für

diese Kommandos speicher muss. Das Dateisystem bleibt flexibel in Hinsicht auf mögliche Erweiterungen.

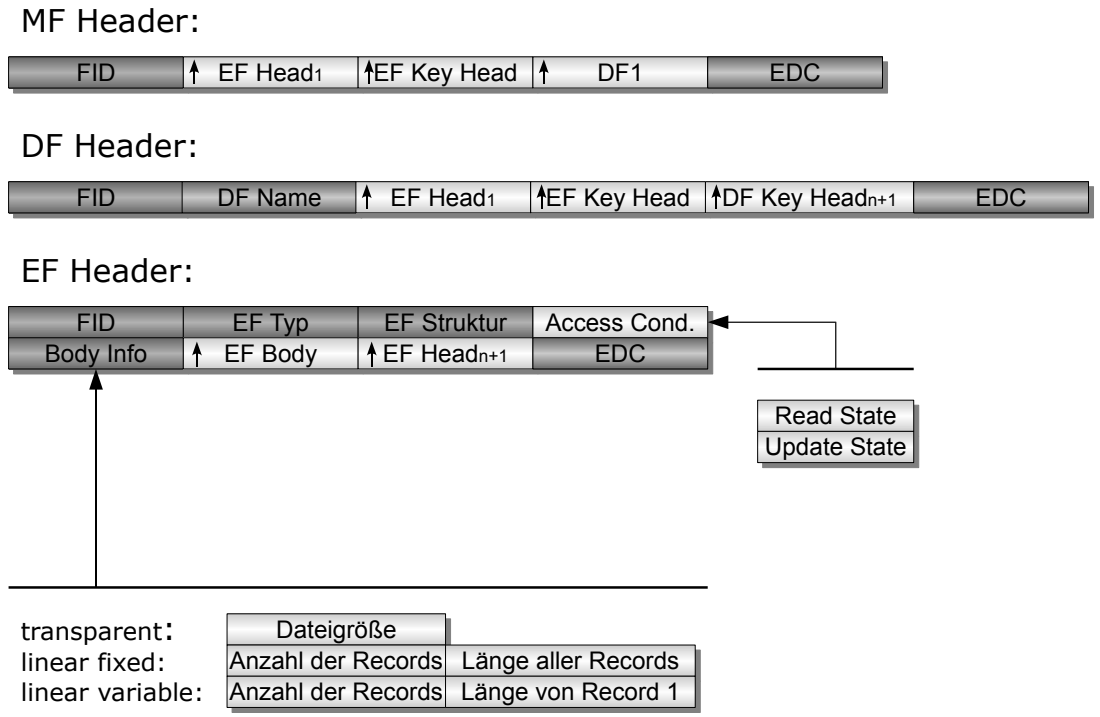


Abbildung 3.20.: Beispielaufbau von Dateiheadern mit TLV-codierten Feldern.

Die Abbildung 3.20 zeigt einen möglichen Aufbau von Dateiheadern, je nach Dateityp, wie es in Rankl (2007) beschrieben ist. Die Felder die dunkel hinterlegt sind, kennzeichnen obligatorische Felder, die hell hinterlegten Felder sind optional. Die Abbildung 3.21 zeigt ein Beispiel von Rankl (2007) wie ein Dateisystem mit Zeigern aufgebaut werden kann.

3.3.3. Dateiverwaltung mittels FAT

Die Dateiverwaltung mittels einer **File Allocation Table** ist schon sehr lange weit verbreitet um Festplatten von PCs zu verwalten. Hierzu wird der Speicher in viele gleich große Stücke aufgeteilt. Am PC ist die Größe dieser Stücke mehr oder weniger wählbar. Je größer diese Stücke sind, desto schneller können große Dateien gelesen und

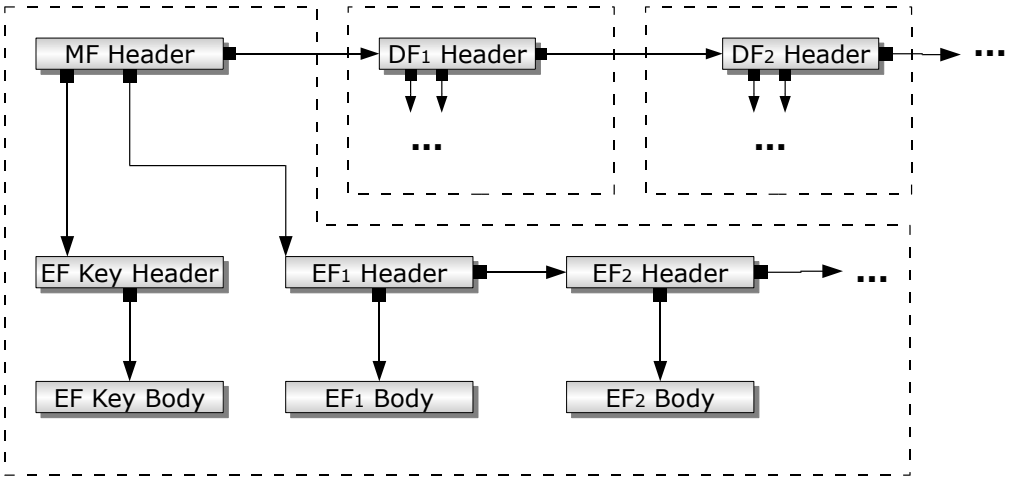


Abbildung 3.21.: Beispielaufbau eines Filesystems mit Zeiger.

geschrieben werden. Das Problem hierbei ist jedoch, dass die Größe von Dateien selten ein Vielfaches der Blockgröße ist. Wenn nun das Ende einer Datei auf einen Block geschrieben wird, welches den Block nicht ganz ausfüllt, ist der Speicher, der auf dem Block nicht verwendet wird, auch nicht für andere Dateien verfügbar. Der Speicher ist somit verschwendet. Wird die Blockgröße zu klein gewählt, wird das Lesen und Schreiben von größeren Daten sehr langsam. Bei Chipkarten wird die Blockgröße optimalerweise an die Größe einer EEPROM Page angepasst.
Jeder einzelne Sektor (EEPROM Page) wird durch einen Zeiger referenziert, welcher in der FAT abgelegt wird. Sollte ein Sektor nicht direkt durch einen FAT Eintrag referenziert werden, da er z.B. Teil einer Datei ist, welche durch einen Zeiger auf einen anderen Sektor referenziert ist, bekommt dieser Sektor in der FAT eine besondere Kennzeichnung. Eine besondere Kennzeichnung bekommen auch defekte Sektoren. Ein Zeiger im Dateideskriptor zeigt also auf eine Position in der FAT, welche wiederum durch einen Zeiger auf den Sektor zeigt, welcher den Anfang der Daten einer Datei beinhaltet. Ist eine Datei nun länger als ein Sektor, zeigt dieser Sektor auf den Sektor, der weitere Daten der Datei enthält. Der Sektor, der die letzten Daten einer Datei beinhaltet, zeigt dann auf EOF, um somit das Ende der Datei anzudeuten. Die Abbildung

3.22 zeigt den grundlegenden Aufbau einer FAT zur Verwaltung von Dateien in einem Chipkarten-Betriebssystem.

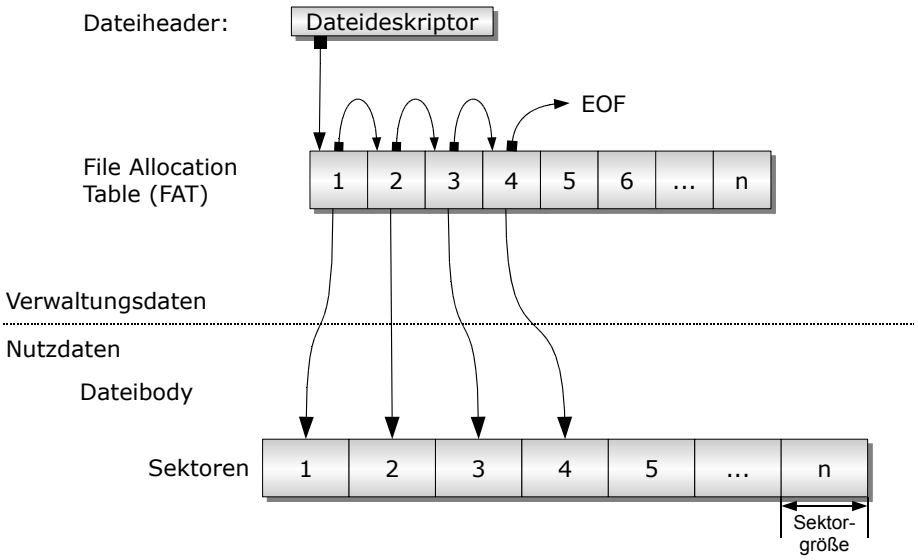


Abbildung 3.22.: Aufbau einer FAT in einem Chipkartenbetriebssystem.

Ob nun eine FAT verwendet werden sollte oder einfach nur eine Referenzierung durch Zeiger vorgenommen werden sollte, hängt vom Verwendungszweck ab. FAT hat nicht nur Vorteile. Wenn ein Chipkartensystem nur wenige kleine Dateien beinhaltet, ist die Realisierung einer FAT nicht sehr sinnvoll, da es im Vergleich zu einem einfachen zeigerunterstützten System enorm viel mehr an Speicher benötigt. Auf der anderen Seite löst sich bei einer FAT das Problem der Fragmentierung sozusagen von selbst, da durch die Organisation durch eine FAT die Speicherdefragmentierung sozusagen automatisch durchgeführt werden kann (oder es kommt erst gar nicht zu einer Speicherfragmentierung).

3.4. Zugriffsbedingungen

Wie schon mehrfach erwähnt, ist die Sicherheit bei Chipkarten ein sehr wesentlicher Punkt. Aus diesem Grund muss es auch möglich sein, den Zugriff auf Dateien zu verwalten. Im Kapitel 3.2.5 wird darauf eingegangen, wie der Zugriff auf Verzeichnisse geregelt ist. Eine Anwendung darf zum Beispiel nicht auf die Dateien einer anderen Anwendung zugreifen. Da Chipkartensysteme eigentlich ausschließlich Einbenutzersysteme sind und es aus diesem Grund keine ganz so komplexen Zugriffskontrollsysteme benötigt, wie es bei manchen Mehrbenutzersystemen von PCs der Fall ist, werden bei Chipkartensystemen die Zugriffsberechtigungen direkt im Header einer Datei abgelegt. Abgesehen von der erwähnten Zugriffskontrolle durch DFs gibt es bei Chipkartensystemen meist noch zwei weitere Möglichkeiten den Zugriff auf Dateien zu definieren, da es durchaus auch sein kann, dass eine Anwendung nur unter bestimmten Bedingungen auf eine Datei zugreifen darf. Abbildung 3.23 zeigt die Klassifizierung der Zustandsbedingungen für den Dateizugriff bei Chipkartensystemen.

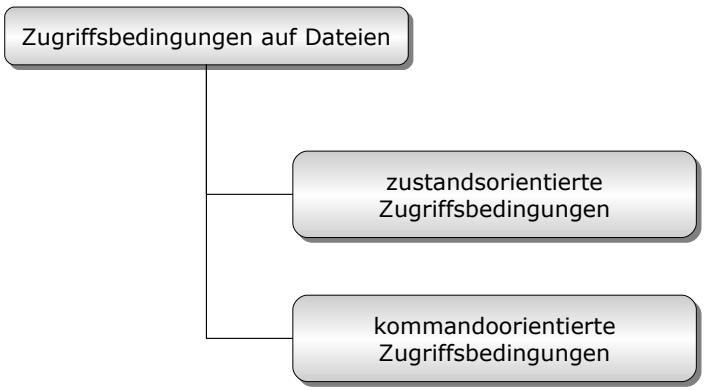


Abbildung 3.23.: klassifizierung der Zustandsbedingungen für den Dateizugriff.

Kommando	Beschreibung
APPEND	Vergrößern einer Datei
CREATE	Erzeugen einer neuen Datei
DELETE FILE	Löschen einer Datei
LOCK	Sperren einer Datei
READ	Lesen aus eine Datei
SEEK	Suchen in einer Datei
WRITE	Schreiben in eine Datei

Tabelle 3.9.: häufig verwendete Kommandos für den Dateizugriff.

3.4.1. Kommandoorientierte Zugriffsbedingungen

Einen Möglichkeit den Zugriff zu kontrollieren ist die Kontrolle abhängig von einem bestimmten Kommando zu machen. Hierfür wird im Header einer Datei eingetragen, welches Kommando Zugriff auf die Datei erhalten soll und welches Kommando nicht. Zudem wird auch die Reihenfolge eingetragen, nach welcher einzelne Kommandos Zugriff auf eine Datei erhalten dürfen. Als Beispiel ist es sinnvoll, ein Kommando wie *'VERIFY PIN'* zu verlangen, bevor ein Kommando wie *'READ'* auf die Datei ausgeführt werden darf. Der Vorteil einer kommandoorientierten Zugriffsbedingung ist sicherlich die Tatsache, dass eine solche Variante recht einfach zu implementieren ist, jedoch mit dem Nachteil, dass sie nicht ganz so flexibel ist wie bei der Verwendung von zustandsorientierten Zugriffsbedingungen. Die Tabelle 3.9 zeigt die am häufigsten verwendeten Zugriffskommandos auf Dateien. Diese Kommandos können aber von Betriebssystem zu Betriebssystem variieren. Nicht jedes Betriebssystem unterstützt die selben Kommandos.

3.4.2. Zustandsorientierte Zugriffsbedingungen

Neben der Möglichkeit im Header einer Datei festzulegen, welche Kommandos der Reihe nach aufgerufen werden müssen, um Zugriff auf die Datei zu erhalten, gibt es auch die Möglichkeit, nach Kommandos die Karte in verschiedene Zustände zu über-

führen. Dabei wird der Zugriff auf Dateien nur dann erlaubt, wenn sich das Chipkarten-Betriebssystem in einem gewissen Zustand befindet.

Zustandsautomaten

Um die zustandsorientierten Zugriffsbedingungen zu implementieren, muss das Chipkarten-Betriebssystem intern einen Zustandsautomaten aufbauen. Ein Zustandsautomat bei Chipkartensystemen ist ausschließlich ein endlicher Automat (finite state machine), der das Verhalten des Chipkartensystemes, die Zustände in welchen sich das System befinden kann, wie auch die Zustandsübergänge beschreibt.
Abbildung 3.24 zeigt einen sehr vereinfachten Zustandsautomaten für eine Geldautomatenabhebung. Die Darstellung ist sehr vereinfacht und ist nicht aus der Sicht des Chipkartensystems sondern mehr aus der Sicht des Benutzers dargestellt.

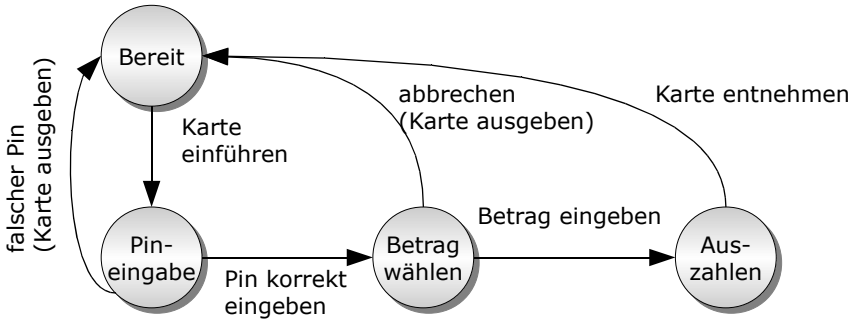


Abbildung 3.24.: Zustandsautomat für einen vereinfachten Geldautomaten.

Der Geldautomat befindet sich, noch bevor ein Benutzer in bedient, in dem Initialzustand **Bereit**. In diesem Zustand zeigt der Bankautomat vielleicht einen Begrüßungsbildschirm an, und intern wird keine Datei zum Lesen freigegeben. Nachdem nun die erste Aktion stattfindet, der Benutzer führt seine Chipkarte in den Leser des Geldautomaten ein, wechselt der Automat vom Zustand **Bereit** in einen neuen Zustand **Pinein-**

gabe. Nun weicht auch der Begrüßungsbildschirm einer Aufforderung an den Benutzer, seine geheime Nummer einzugeben. Intern läßt der Automat immer noch keine Zugriffe auf Dateien zu, da in dem neuen Status immer noch keine Überprüfung der Berechtigung durchgeführt wurde. Gibt nun der Benutzer die falsche PIN ein, wechselt der Automat wieder zurück in den Initialzustand **Bereit** und gibt die Karte wieder aus. Nun kann der Benutzer noch einmal die Karte einführen und einen weiteren Versuch zur Pineingabe starten. Gibt der Benutzer aber die korrekte PIN ein, wechselt der Automat in den neuen Zustand **Betrag wählen**. In diesem Zustand können vom Benutzer schon Daten gelesen werden, da er sich durch die PIN erfolgreich authentifiziert hat. Nun hat der Nutzer die Möglichkeit einen Betrag zur Auszahlung zu wählen, oder aber er kann ohne Auszahlung die Karte zurückfordern. Fordert der Benutzer seine Karte ohne Auszahlung zurück, gibt der Automat die Karte dem Benutzer zurück und wechselt wieder in den Initialstatus **Bereit**. Wählt der Benutzer einen Betrag zur Auszahlung wechselt der Automat in den Zustand **Auszahlen** (positive Deckungsprüfung vorausgesetzt). In diesem Zustand hat das System schon alle Dateien wieder gesperrt und wartet nur noch, dass der Benutzer die Karte wieder an sich nimmt. Ist dies erfolgt, wechselt der Automat zurück in den Initialzustand **Bereit** und wartet auf neue Aufträge.

3.5. Zugriffsbedingungen nach ISO/IEC7816-9

Ob man sich beim Betriebssystemdesign für kommandoorientierte Zugriffsbedingungen oder für die zustandsorientierten Zugriffsbedingungen entscheidet ist mehr oder weniger Geschmacksache. Aus diesem Grund gibt es beide Varianten in verschiedenen auf dem Markt befindlichen Betriebssystemen. Durch die Vielfalt der unterschiedlichen Betriebssysteme auf dem Markt unterscheiden sich auch die Implementationsformen und Lösungsansätze der Zugriffsbedingungen. Um diesem Problem Herr zu werden, definiert die ISO/IEC 7816-9 (2004) die Zugriffe auf Ressourcen eine Chipkarte. Abbildung 3.25 zeigt die nach ISO/IEC 7816-9 (2004) definierten Zugriffsbedingungen für Kommandos und Dateien.

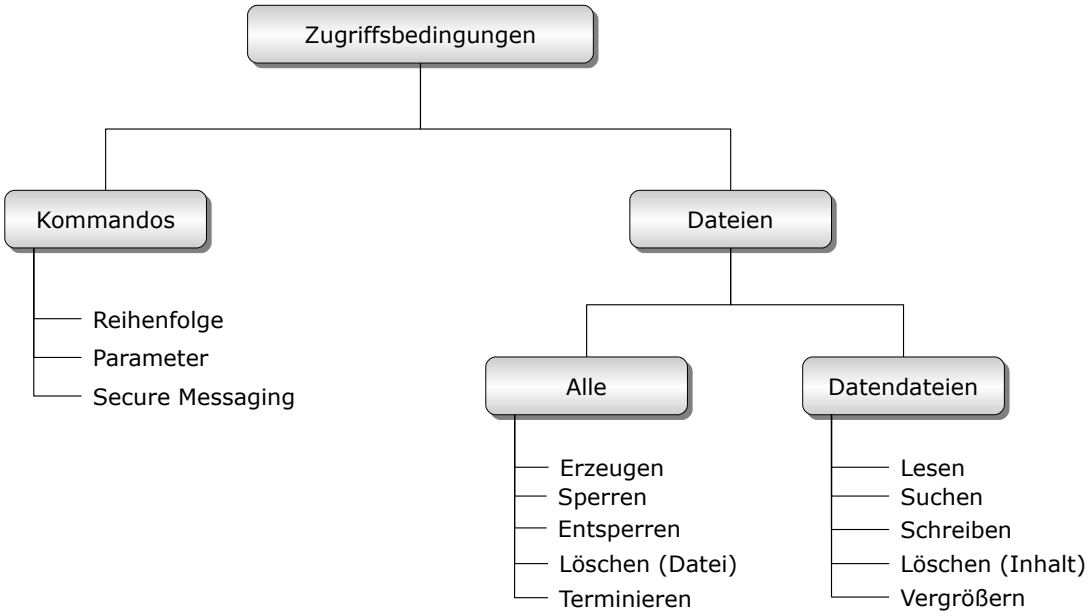


Abbildung 3.25.: Zugriffsbedingungen nach ISO/IEC 7816-9 (2004) für Kommandos und Dateien.

Das Modell nach ISO/IEC 7816-9 (2004) baut auf TLV codierten Datenobjekten auf und vereint zustands- wie auch kommandoorientierte Zugriffsbedingungen. Zusätzlich gibt es in diesem Modell noch die Möglichkeit, bestimmte Kommandosequenzen festzule-

gen. Es werden **Sicherheitsattribute (security attributes - SA)** definiert, die sowohl den Zugriff, als auch den Nicht-Zugriff regeln. Diese SA lassen sich auch dazu verwenden um bestimmte Sicherheitszustände zu erreichen. Sie steuern den Zugriff auf Ressourcen, wie Kommandos, Dateien aber auch Datenobjekte. Abbildung 3.26 zeigt die möglichen Kartenressourcen mit Sicherheitsattributen.

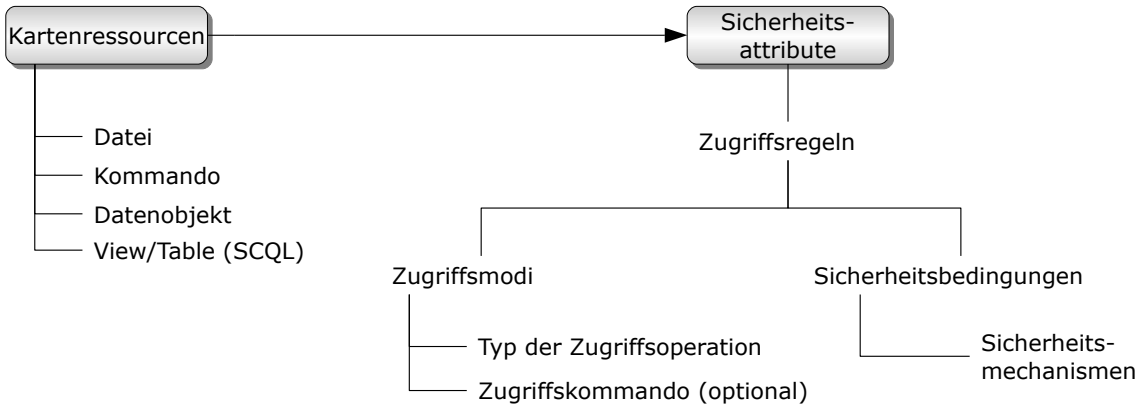


Abbildung 3.26.: Klassifizierung der Kartenressourcen und Sicherheitsattribute nach ISO/IEC 7816-9 (2004).

Jede Ressource besitzt einen Zeiger auf ein oder auch mehrere Sicherheitsattribute. Diese wiederum bestehen aus **Zugriffsregeln** welche sich aus **Zugriffsmodi (access mode - AM)** und **Sicherheitsbedingungen (security conditions - SC)** zusammensetzen. Der Zugriffsmodus legt fest, ob auf die Datei zum Beispiel lesend oder schreibend zugegriffen wird und die Sicherheitsbedingungen definieren die **Sicherheitsmechanismen (security mechanism - SM)**. ISO/IEC 7816-9 (2004) legt fest, dass die Sicherheitsattribute speicherplatzsparend in TLV codierten Datenobjekten gespeichert werden. Zusätzlich wäre es auch möglich die Sicherheitsattribute in einem Compact-

Format zu speichern. Die Zugriffsregeln werden in EFs gespeichert, welche die Struktur *linear variable* besitzen. Die Datei wird als EF_{ARR} benannt. Der FID kann jedoch frei gewählt werden. Die Abbildung 3.29 zeigt den Header einer Datei mit Zeiger auf eine Datei, welche die Zugriffsregeln beinhaltet.

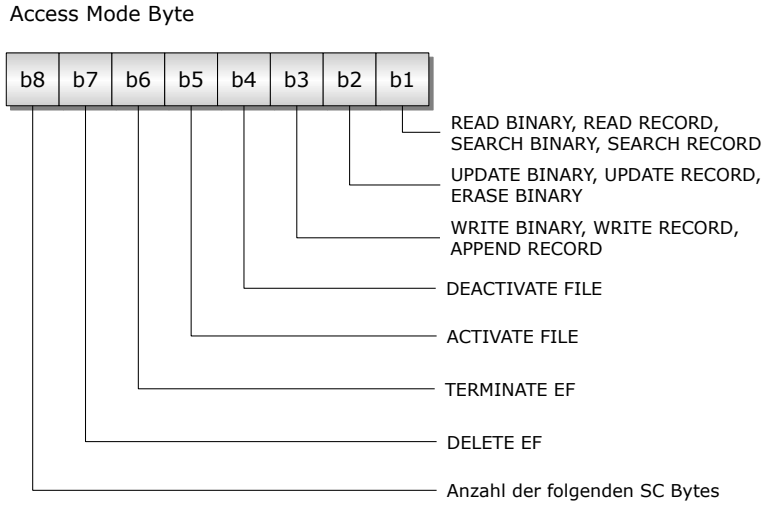


Abbildung 3.27.: Belegung des Access Mode Bytes.

Wenn nun auf eine Datei zugegriffen werden soll, definiert ISO/IEC 7816-9 (2004) den folgenden Ablauf. Der erste Schritt ist die Prüfung auf das Vorhandensein einer Referenz auf ein EF ARR. Ist keine solche Referenz vorhanden, wird der Zugriff auf die Datei verweigert. Ist eine Referenz auf ein EF ARR vorhanden, wird geprüft, ob der referierende Record auch im EF ARR eingetragen ist. Ist das nicht der Fall, wird der Zugriff verweigert. Wird der Record gefunden, wird geprüft, ob ein entsprechender Zugriffsmodus für den geforderten Zugriff existiert. Sollte dies nicht der Fall sein, wird abgebrochen und der Zugriff verweigert. Sollte ein passender Zugriffsmodus gefunden werden, wird geprüft ob für diesen die Sicherheitsbedingung erfüllt ist. Ist die Sicherheitsbedingung für den geforderten Zugriffsmodus erfüllt, wird der Zugriff auf die Datei gestattet. Sollte die Bedingung nicht passen, wird der Zugriff verweigert. Die Abbildung 3.30 zeigt diesen Vorgang in Form eines Flußdiagramms.

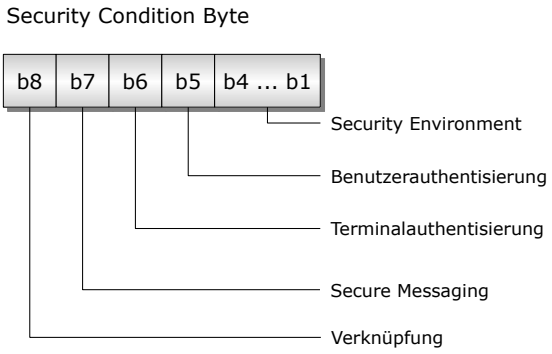


Abbildung 3.28.: Belegung des Security Condition Bytes.

3.6. Speicherverwaltung

Neben dem Aufbau des Dateisystems muss sich ein Betriebssystem für Chipkarten auch um die Verwaltung des Speichers kümmern. Dazu gehören auch Dienste, die für die Anforderung und Freigabe von Speicher zuständig sind. Ein Programm hat dazu mehrere Anforderungen an das System. Dazu gehören unter anderem *Speicher anfordern*, *Speicher vergrößern*, *Daten lesen*, *Daten speichern* und *Daten atomar speichern*. Auf diese noch sehr grundlegenden Operationen baut eine Schnittstelle des Dateimanagements auf, welche dann konkretere Operationen zur Verfügung stellt. Diese sind beispielsweise Dienste wie **Datei anlegen**, **Datei löschen**, **Daten lesen** (aus einer Datei), **Daten schreiben**. Diese Funktionalität sollte jedes Betriebssytem unterstützen, nicht nur Betriebssysteme von Chipkarten. Daneben gibt es noch Funktionen, die schon spezieller für Chipkartensysteme sind, wie **MF selektieren**, **übergeordnetes DF selektieren**, **Datei mittels DF Name selektieren** und **Datei durch FID selektieren**.

3.6.1. Aufteilung des Speichers in Speicherseiten

Zwei wichtige Aspekte der Architektur einer Chipkarte beeinflussen sehr stark das Speichermanagement eines Chipkarten-Betriebssystems. Auf der einen Seite die beschränkte Anzahl von Lese- und Schreibzugriffen auf das EEPROM. Daher sollte das

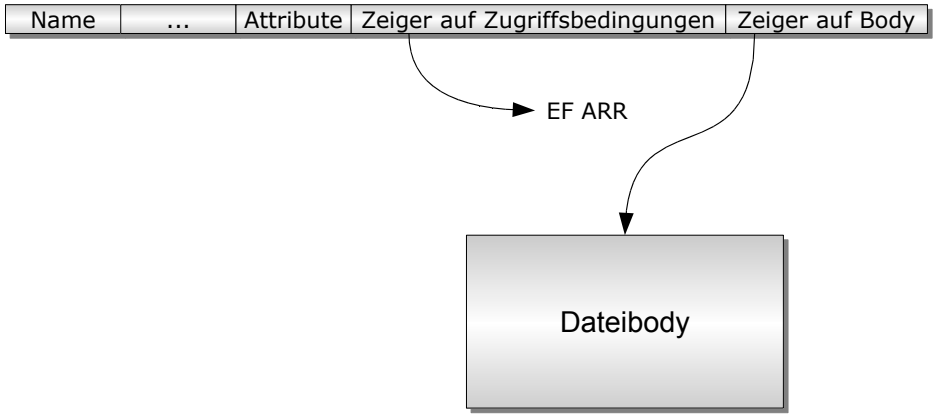


Abbildung 3.29.: Header einer Datei mit Zeiger auf Zugriffsrechtedatei.

Betriebssystem versuchen so wenig wie möglich Lese- und Schreibzugriffe auf das EEPROM durchzuführen. Auf der anderen Seite beeinflulsst die Einteilung des EEPROMs in Speicherseiten das Betriebssystemdesign. Aus diesem Grund ergibt sich die Tatsache, dass bei Chipkarten-Betriebssystemen die Dateiheader (Verwaltungsdaten) physikalisch streng getrennt von den Dateibodies (Nutzdaten) gespeichert werden. Wird diese Trennung nicht durchgeführt, kann es zu problematischen Seiteneffekten kommen, wodurch sich zum Beispiel durch die ungewollte (oder auch gewollte) unerlaubte Überschreibung des Headers einer Datei der Zustand ergeben kann, dass die Zugriffsbedingungen gelöscht werden und somit geheime Daten plötzlich zugänglich gemacht werden.

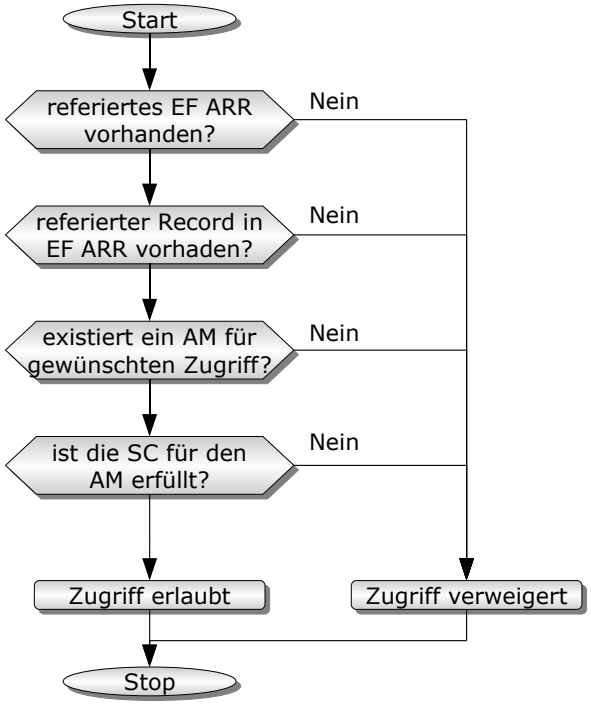


Abbildung 3.30.: Prüfung bei Zugriff auf eine Datei nach ISO/IEC 7816-9 (2004).

3.6.2. Aufteilung von Dateien durch FDs

Eine weitere Sicherheitsmaßnahme, die den unerlaubten Zugriff auf Daten verhindern soll, ist die Separierung von Dateien durch EFs. Wie schon in Kapitel 3.2.5 erwähnt, werden Dateien einer Anwendung direkt unter das DF der jeweiligen Anwendung gespeichert. Aufgabe des Betriebssystems ist es dafür zu sorgen, dass unter keinen Umständen andere Anwendungen auf die Inhalte eines DFs zugreifen können. Die meisten Chipkarten (Infineon bietet auch schon Karten mit MMU an) bieten hierfür noch keine MMU (Memory Management Unit), welche die Aufgabe hardwareseitig übernimmt, somit muss diese Aufgabe noch oft vom Betriebssystem übernommen werden.

3.6.3. Mechanismen zur Freispeicherverwaltung

Erst seit Mitter der 90er Jahre bieten Betriebssysteme die Möglichkeit, Dateien auch nach der Personalisierung einer Chipkarte zu erstellen oder auch zu löschen. Davor wurden alle im späteren Betrieb benötigten Dateien bei der Personalisierung einer Chipkarte angelegt und blieben auf der Karte gespeichert, bis die Karte vernichtet wird. Lediglich die Dateiinhalte konnten geändert werden. Modernere Systeme bieten heute die Möglichkeit auch Dateien nach der Personalisierung wieder zu entfernen und neue zu erstellen. Dies darf selbstredend nur unter Verwendung kryptographischer Sicherungsmechanismen erfolgen. Neben der Zugriffssicherung muss ein System bei der Freispeicherverwaltung auch die spezielle Art der Verwendung einer Chipkarte berücksichtigen. Es kann ohne weiteres vorkommen, dass eine Chipkarte plötzlich aus dem Terminal entfernt wird und den Strom verliert. Wenn die Chipkarte zu diesem Zeitpunkt gerade einen Schreibzugriff durchgeführt hat, darf es durch das Entfernen der Chipkarte keinesfalls zu einem undefinierten Zustand kommen. Das Betriebssystem muss also Schreiboperationen **atomar** durchführen. Von einer **atomaren Transaktion** spricht man, wenn eine Transaktion nicht unterbrochen werden kann. Das heißt, die Transaktion wird komplett durchgeführt oder sie wird überhaupt nicht durchgeführt. Um dies zu gewährleisten, wird der Status der Transaktion protokolliert. Sollte die Transaktion nicht durchgeführt werden können, wird sie mit Hilfe der protokollierten Daten rückgängig gemacht. Im Buch Kemper (2004), Kapitel 9.5 (Eigenschaften von Transaktionen) ist die atomare Eigenschaft von Transaktionen detailliert beschrieben.
Bei Chipkarten haben sich mehr oder weniger drei Mechanismen zur Freispeicherverwaltung als vorteilhaft erwiesen.

WORM

Der Mechanismus des **Write Once Read Multiple** ist der am einfachsten zu realisierende. Die Datei wird einmal angelegt und kann dann so oft wie notwendig gelesen werden. Wird die Datei (logisch) gelöscht, wird der Speicherplatz weiterhin von der Datei belegt. Abbildung 3.31 zeigt diesen Mechanismus.

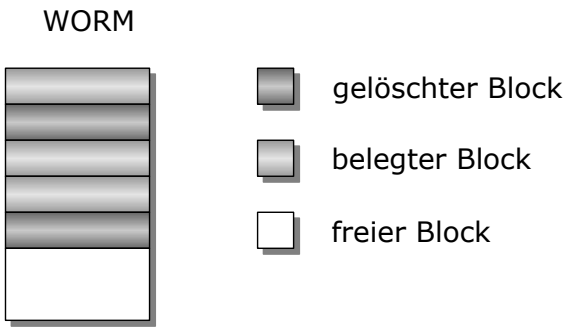


Abbildung 3.31.: Mechanismus zur Speicherverwaltung - Write Once Read Multiple.

LIFO

Beim **Last In First Out** Verfahren kann jeweils nur die zuletzt geschriebene Datei gelöscht werden. Dieses Verfahren ist schon ein wenig aufwändiger zu implementieren als das WORM Verfahren. Da es noch sehr einfach ist, aber dennoch effizient findet, es sich oft in Chipkarten-Betriebssystemen.

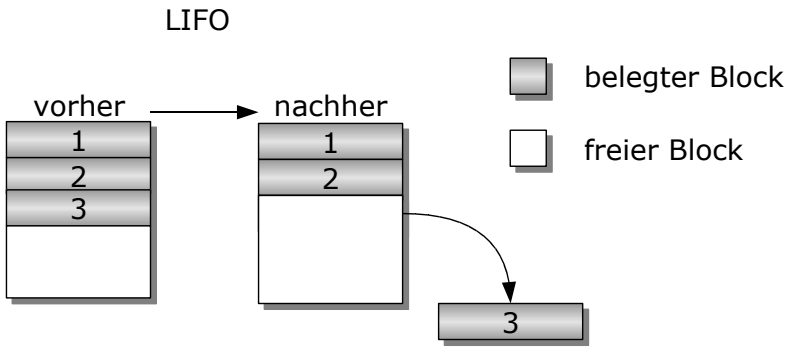


Abbildung 3.32.: Mechanismus zur Speicherverwaltung - Last In First Out.

best-fit

Beim schon etwas komplizierteren **best-fit** Verfahren sucht sich das Betriebssystem den am besten passenden Speicherplatz für das Datenelement aus. Hier wird gezielt darauf geachtet, dass es nur zur geringst möglichen Fragmentierung kommt. Wenn aber doch viele Dateien gelöscht und auch wieder erzeugt werden, läßt es sich nicht vermeiden, dass eine gewisse Fragmentierung entseht. Der Grund hierfür ist einfach. Da fast jede Datei unterschiedlich groß ist, kommt es vor, dass zu kleine Speicherblöcke, die zwar frei sind, nicht genutzt werden können, da die neu zu erstellende Datei einen größeren Speicherblock benötigt. Wenn der Speicher zu sehr fragmentiert ist, kann es dadurch vorkommen, dass plötzlich keine größeren Dateien mehr angelegt werden können.

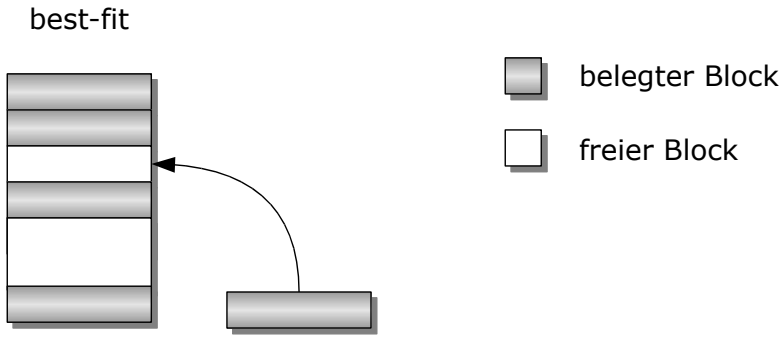


Abbildung 3.33.: Mechanismus zur Speicherverwaltung - Best-Fit.

Defragmentierung

Ein spezieller Mechanismus des Chipkarten-Betriebssystems kann dafür sorgen, dass die Fragmentierung nicht zu stark wird, in dem der Speicher **defragmentiert** wird. Hier wird versucht, belegte Blöcke zusammenhängend im Speicher zu platzieren, ebenso wie auch nicht belegt Blöcke. Abbildung 3.34 zeigt diesen Vorgang.

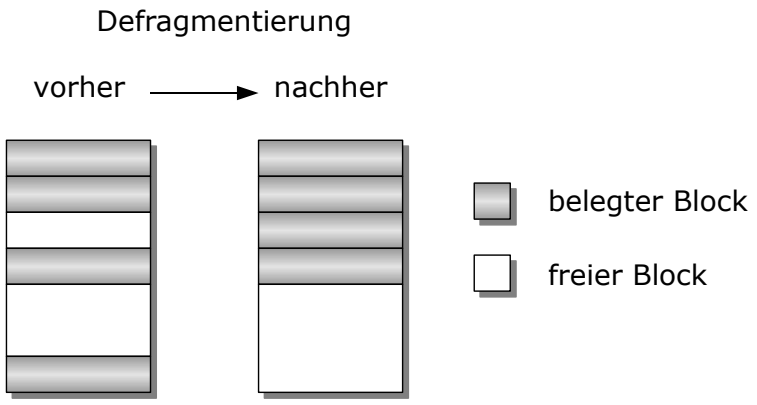


Abbildung 3.34.: Mechanismus zur Speicherverwaltung - Defragmentierung.

Garbage Collection

Der Mechanismus der **Garbage Collection** kann bei allen oben beschriebenen Verfahren zum Einsatz kommen. Ein Gragabe Collection Mechanismus durchsucht in gewissen Zeitabständen den gesamten Speicher nach nicht mehr benötigten Blöcken. Wird er fündig, wird der Block wieder als freigegeben markiert und kann wieder verwendet werden. Abbildung 3.35 zeigt diesen Mechanismus.

Datenintegrität

Bei Chipkarten ist die Datenintegrität überaus wichtig. Aus diesem Grund sollte ein Betriebssystem auch einen Mechanismus vorsehen, der die Datenintegrität sicherstellt. Um sicherzustellen, dass Daten nicht unerlaubt verändert wurden, bietet sich der Einsatz von Checksummen an. Bei Chipkarten kommt oft das CRC Verfahren zum Einsatz, da es schnell zu berechnen und nicht sehr komplex zu implementieren ist.

Ein besseres Verfahren, speziell auch in Hinblick auf die Ausfallscharakteristik von EEPROMs, ist der Reed-Solomon-Code. Dieses Verfahren kommt auch bei anderen An-

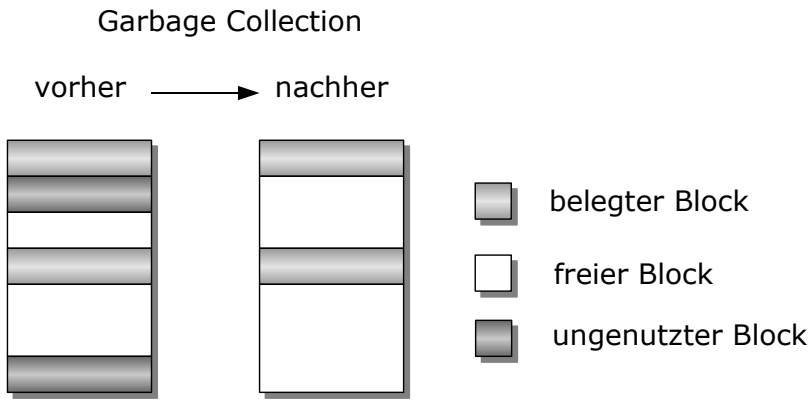


Abbildung 3.35.: Mechanismus zur Speicherverwaltung - Garbage -Collection.

wendungen zum Einsatz. Beispiele hierfür sind bei der Datenspeicherung Barcodes, CDs oder DVDs und bei der Datenübertragung DSL wie auch Satelliten. Mehr zu diesem Verfahren beschreibt Rankl (2007) in seinem Buch.

4. Conclusio

Die Geschichte zeigt eine rasante Entwicklung im Bereich der Chipkarten. Es werden laufend neue Anwendungen gefunden, die Möglichkeiten sind enorm. Die vorliegende Arbeit gibt dem Leser eine Einführung in die Internas von Chipkarten-Betriebssystemen. Der Focus ist klar auf die Dateiverwaltung wie auch die Speicherverwaltung gesetzt. Um zu zeigen, dass Chipkartenentwickler das Rad nicht neu erfunden haben, wird das Dateisystem FAT der Firma Microsoft detailliert behandelt. Hier finden sich einige Parallelen zu möglichen Dateisystemen, wie sie in Chipkarten eingesetzt werden können. Ebenso finden sich auch einige Unterschiede wieder, da die Entwicklung eines Systems für Chipkarten doch ganz andere Anforderungen hat, als es beim PC der Fall ist. Bei Chipkarten spielt der Speicher und auch die Ausführungsgeschwindigkeit, wie auch Sicherheit eine wesentliche Rolle. Bei PC Systemen ist die Sicherheit ganz bestimmt auch nicht vernachlässigbar, jedoch sind die Ressourcen doch nicht annähernd so beschränkt wie bei Kartensystemen.

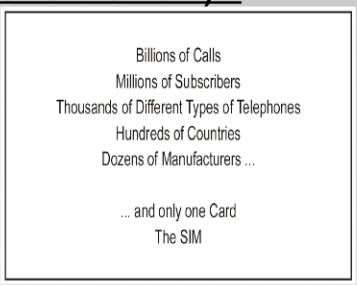
Ziel der Arbeit ist es, eine Einführung zu geben, wobei der Focus auf die Dateiverwaltung gesetzt wurde. Daher sind auch andere wesentliche Gebiete der Chipkarten-Betriebssysteme leider unberücksichtigt geblieben, da sonst der Rahmen dieser Arbeit um ein Vielfaches gesprengt worden wäre. Im Anhang finden sich zahlreiche Referenzen, welche die nicht behandelten Gebiete umfassend abdecken.

A. Unterichtsunterlagen

Warum Chipkarten (Hardware-Token)?

- Aus dem großen Bedarf nach einer sicheren Identifikation, die eine sichere Authentifikation erfordert ➔ Chipkarten wie SIM-Karten, Bankkarten, Unternehmenskarten etc.

  Chipkarte erlaubt mehrstufige Authentifikation (z.B. PIN oder Biometrie vom Benutzer und Challenge&Response zum IT-System)
- Aus dem zunehmenden Bedarf nach der Digitalen Signatur, die eine hochsichere Signaturerstellungseinheit erfordert



Hardware Token, Markus Winkler 1

Warum Chipkarten (Hardware-Token)?

- Aus dem großen Bedarf nach sicherer Rechenleistung vor Ort auf einem Kleinstcomputer wie für die Erzeugung von MACs und der Digitalen Signatur, für elektronische Geldbörsen (z.B. Quick in Österreich), für Schlüsselableitungen (z.B. bei SIM-Karten), zur Erzeugung von Zufallszahlen und zur Datenverschlüsselung (für Challenge&Response Authentifizierung, für verschlüsselte Datenübertragung etc.), zur Überprüfung von biometrischen Daten etc.

Hardware Token, Markus Winkler 2

Warum Chipkarten (Hardware-Token)?

- Aus dem großen Bedarf nach geschützten Daten vor Ort wie kryptografischen Daten (Schlüssel für Datenverschlüsselung und Digitale Signatur, für Schlüsselableitungen, für Authentifikation, PayTV-Schlüssel etc.), Karteninhaberdaten (medizinische Daten, biometrische Daten etc.), Zahlungsverkehrsdaten (Geldbörsenbetrag, Kontodaten, Punkte etc.), Berechtigungsdaten für Offline-Berechtigungskontrolle (Zutritts- und Zugriffsberechtigung, Benutzungskontrolle (z.B. Startschlüssel), Benutzerprofil, etc.), Tickets (Einzel/Mehrfach, Zeitkarten), sonstige Daten

Hardware Token, Markus Winkler 3

Warum Chipkarten (Hardware-Token)?

Für Spezialanwendungen:

- für den hochsicheren Transport von kryptografischen Schlüsseln (meist aufgeteilt auf mehrere Chipkarten)
- für lokale Anwendungen für Mobiltelefone auf der SIM-Karte
- für den Softwareschutz (passive oder aktive kartenbasierende Dongle)
- etc.

Hardware Token, Markus Winkler 4

Klassische Chipkarten

- SIM-Karte, USIM-Karte für mobile Telekomm.
- Bankkarte: Kreditkarte, Debitkarte, Geldbörse, ...
- Versichertenkarten (z.B. e-card)
- Öffentlicher Nahverkehr: Ticket, Zeitkarte, ...
- Pay-TV Card
- ID-Karten wie Personalausweis, Reisepass, ...
- Unternehmenskarte, Bürgerkarte, ...
- Berechtigungskarten: Startschlüssel, Türöffner, ...
- Sonstige Karten wie Digital Tacho (Fahrtenschreiber), Handelskarten, Tourismuskarten, Notfallkarten etc.

Hardware Token, Markus Winkler 5

Was zählen wir zu Chipkarten (Hardware-Token)?

Wir zählen alle

Hardware-Token mit Chipkartenfunktionalität zu Karten, weil es uns um die Funktionalität geht, d.h. neben allen Chipkarten auch Reisepass, USB-Gerät und RFID-Tag mit Chipkartenfunktionalität, Startschlüssel, ...

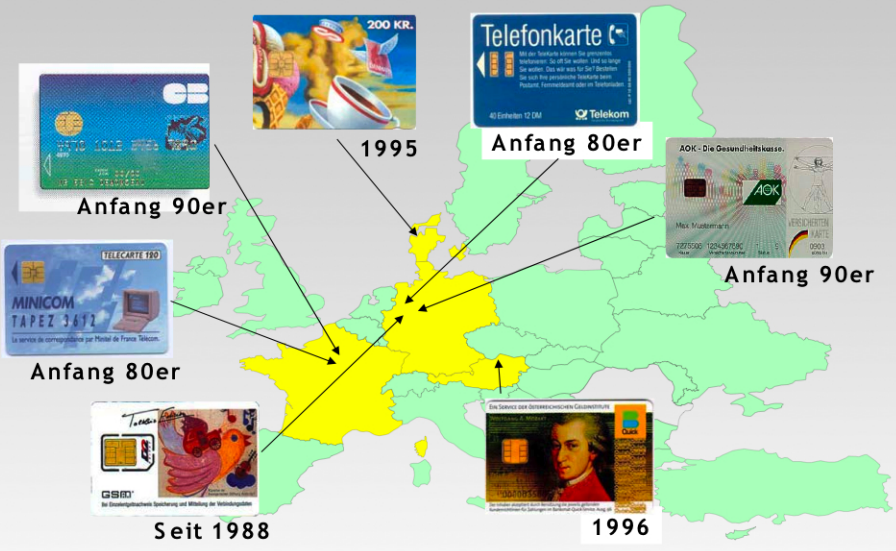
Hardware Token, Markus Winkler 6

Geschichte der Chipkarten

- 1968 Patent Jürgen Dethloff (Deutschland)
- 1970 Patent Arimura (Japan)
- 1974 Patent Moreno (Frankreich)
- 1984 Frankreich: Telefonkarten mit Chip in preisgünstiger EPROM-Technologie
- 1987 Kreditkarten in Frankreich erhalten Chip
- 1991 GSM SIM-Karten (ab 1992 erste Länder)
- 1996 Weltweit erste landesweite multifunktionale Bankkarte mit Chip (Österreich, Patent Piller)
- 1996: Erste landesweite SV-Karte (Deutschland)

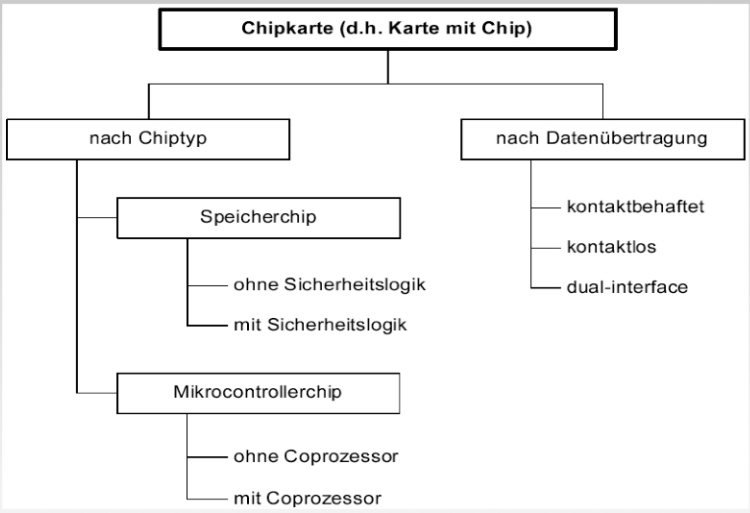
Hardware Token, Markus Winkler 7

Chipkarte: Start und Ausdehnung in Europa



Hardware Token, Markus Winkler 8

Chipkarten Klassifizierung



* Abbildung nach Rankl W., Handbuch der Chipkarten, Hanser Verlag 2002

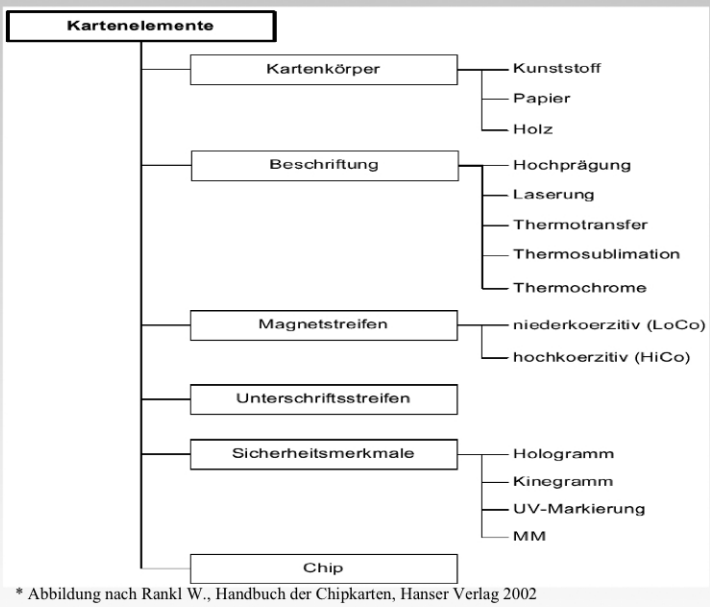
Hardware Token, Markus Winkler 9

Chipkartenformate

- ID-1 Format
- ID-000 Format
- ID-00 Format
- Andere Formate:
  - Reisepass
  - Größe von Memory Sticks
  - Autoschlüssel
  - Uhr
  - etc.

Hardware Token, Markus Winkler 10

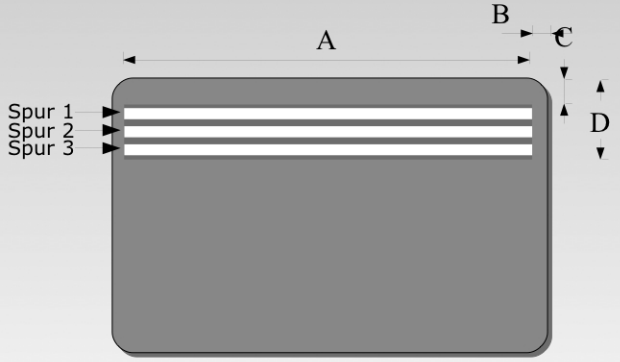
Elemente einer Chipkarte



* Abbildung nach Rankl W., Handbuch der Chipkarten, Hanser Verlag 2002

Hardware Token, Markus Winkler

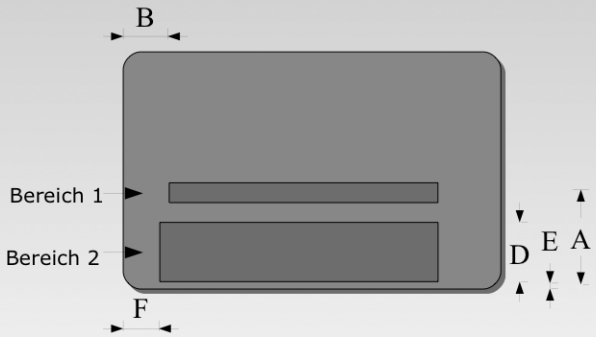
Lage des Magnetstreifens



A: 82,55 mm
B: 2,92 mm
C: 5,54 mm
D: 15,82 mm

Hardware Token, Markus Winkler 12

Lage des Hochprägung



A: 21.42 mm ± 0,12mm
B: 10,18 mm ± 0,25mm
D: 14,53 mm
E: 2.41 mm ... 3,30mm
F: 7,65 mm ± 0,25mm

Hardware Token, Markus Winkler 13

Sicherheitsmerkmale der Karte

- Unterschriftsstreifen : heißverklebt
- Guiochen: verwobene Linienfelder
- Mikroschrift: nicht kopierbar
- UV-Schrift
- Hologramm: integriert
  - Prägehologramm /Weißlichthologramm
  - Im „Laminier-“ oder „Roll on Verfahren“
  - Hot stamp Verfahren

Hardware Token, Markus Winkler 14

Sicherheitsmerkmale der Karte

- Kinegramm, Kippbild
- Multiple Laser Image : kleines Bild
- Lasergravur
  - Vektorverfahren: 1 Sekunde
  - Rasterverfahren: 10 Sekunden
  - Mit und ohne Deckfolie
- Hochprägung

Hardware Token, Markus Winkler 15

Klassifizierung von Chipkarten

Nach Schnittstelle nach Außen:

- Kontaktbehaftet
- Kontaktlos
- Dual-Interface oder Hybrid-Chip (2 Chips)

Nach Chip-Leistung:

- Speicherchip
- Mikrocontrollerchip
- Mikrocontrollerchip, Co-Prozessor, ….

Hardware Token, Markus Winkler 16

Speicher-Chipkarte

- Hauptanwendung als Telefonkarte:
  - Verwendung von nichtlöschbarem Speicher
  - Reduzierung verbrauchter Einheiten irreversibel
  - Gespeicherte Werte können nicht erhöht werden
- Sicherheitslogik verhindert Angriffe von Außen
- Über einzigartige Seriennummer lassen sich von Außen spezielle Sicherheitssysteme realisieren
- Zugriffschutz und Veränderungsschutz von einzelnen Datenfeldern
- Ein oder mehrere Passwörter
- Chippreis und -fläche gering (wenige mm²)
- Genutzt als Wertkarte, Ausweiskarte, Datenträger

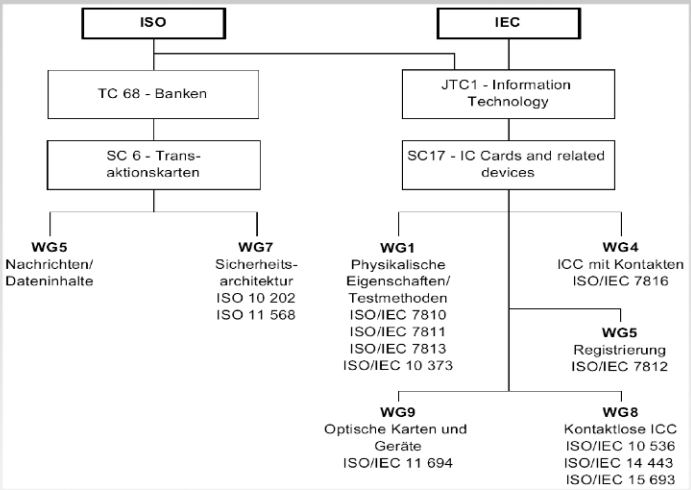
Hardware Token, Markus Winkler 17

Mikroprozessor-Chipkarten

- Karte frei programmierbar
- Beschränkt durch Rechenleistung und Speicher
- Nachladbarkeit von Applikationen
- Kann Krypto-Algorithmen rechnen (meist über Coprozessor)
- Potenzial noch nicht ausgeschöpft

Hardware Token, Markus Winkler 18

Chipkarten Standards



* Abbildung nach Rankl W., Handbuch der Chipkarten, Hanser Verlag 2002

Hardware Token, Markus Winkler 19

Chipkarten Standard ISO 7816

ISO/IEC 7816 ist die wichtigste weltweite Norm für Chipkarten

- Definiert die Karte
- Definiert die Schnittstelle nach Außen
- Definiert alle grundlegenden Eigenschaften von Chipkartenbetriebssystemen und der dazugehörenden Informationstechnik
- Definiert die Kryptografie, Biometrie, ……

Hardware Token, Markus Winkler 20

ISO 7816-1: Physical characteristics

- Formate (Größe, Biegbarkeit)
- Bezieht sich auf ISO 7810, 7811, 7812 und 7813
- Tests

ISO 7816-2: Dimensions and location of the contacts

ISO 7816-3: Electronic signals and transmission protocols

- Spannungs- und Stromversorgung
- Taktfrequenzen, Resetverhalten
- ATR, PTS
- Protokolle T=0, T=1

Hardware Token, Markus Winkler 21

ISO 7816-4: Interindustry commands for interchange

- Daten und Dateien
- Kommandos, Return Codes
- Secure Messaging (zumindest teilweise)
- Logical Channels

ISO 7816-5: Numbering system and registration procedure for application identifiers

ISO 7816-6: Interindustry data elements

- Datenelemente (Tags; DO – Data Objects)

ISO 7816-7: Interindustry commands for Structured Card Language (SCQL)

- Selten eingesetzte Funktionalität, entspricht SQL

Hardware Token, Markus Winkler 22

ISO 7816-8: Security related interindustry commands

- Definiert sicherheitsrelevante Funktionen und Kommandos

ISO 7816-9: Additional interindustry commands and security attributes

- Beschreibt den Kartenlebenszyklus
- Zugriffskontrolle
- Suchkommandos für Dateninhalte
- Erzeugung und Löschen von Dateien

ISO 7816-10: Electronic signals and answer to reset for synchronous cards

Hardware Token, Markus Winkler 23

ISO 7816-11: Personal verification through biometric methods

ISO 7816-12: USB electrical interface and operating procedure

ISO 7816-13: Registration of integrated circuit manufacturers

ISO 7816-15: Cryptographic information application

- Entspricht dem Standard PKCS#15, jedoch mit Ausrichtung auf Chipkarten

Hardware Token, Markus Winkler 24

Hauptaufgaben von Chipkartenbetriebssystemen

- Datenübertragung von und zur Chipkarte
- Ablaufsteuerung der Kommandos
- Dateiverwaltung
- Zustandsautomat zur Berechtigungssteuerung
- Sicherheitssystem
- Verwaltung und Ausführung von kryptografischen Algorithmen
- Verwaltung und Ausführung von Anwendungssoftware
- Standardisiert nach ISO 7816

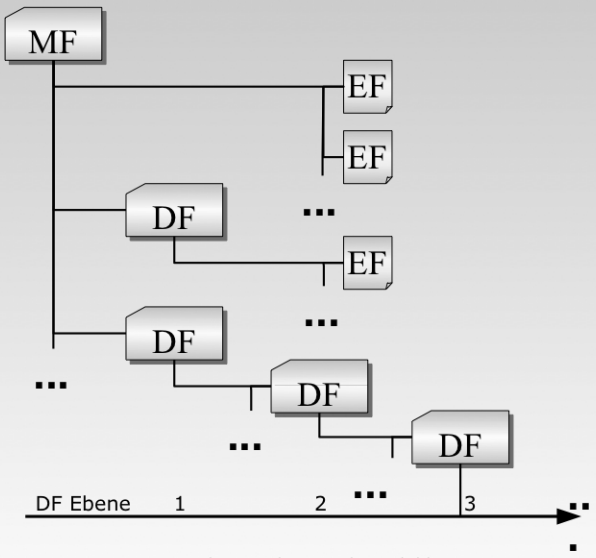
Hardware Token, Markus Winkler 25

Dateisystem und Security nach ISO 7816

- ISO 7816-4 definiert ein Chipkarten-Dateisystem
- Es ist hierarchisch strukturiert mit Verzeichnissen und Dateien
- Aufbau:
  - MF Master File: Höchste Ebene des Dateisystems
  - DF Dedicated File: Verzeichnis mit Referenzen zu DFs und EFs (Elementary Files)
  - ADF Enthält alle Dateien zu einer spezifischen Anwendung
  - EF Elementary File: Datei mit den Daten
- Identifikation erfolgt über 2-Byte Adressen

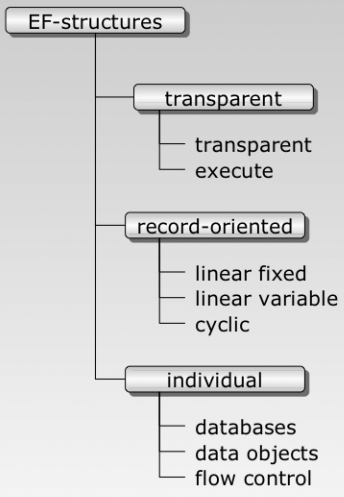
Hardware Token, Markus Winkler 26

Dateibaum ISO 7816



Hardware Token, Markus Winkler 27

EF-Strukturen



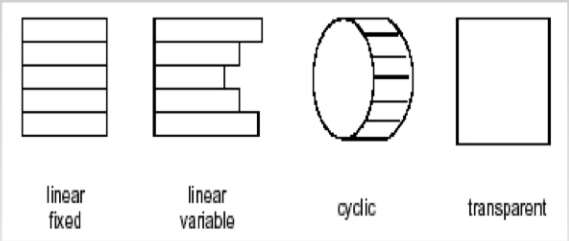
Hardware Token, Markus Winkler 28

Dateistrukturen

- Transparent
  - Direktzugriff zu den Daten
  - Lese-/Schreib-Zugriff benötigt Dateiname, Offset in die Datei und Länge
- Fixed Record: Sequentielle Datei mit fester Satz-Länge
- Variable Record: Sequentielle Datei variabler Satz-Länge
- Cyclic
  - Sequentielle Datei mit fester Satz-Länge
  - Schreibt immer in den nächsten Satz
  - Hinter dem letzten Satz befindet sich wieder der erste Satz

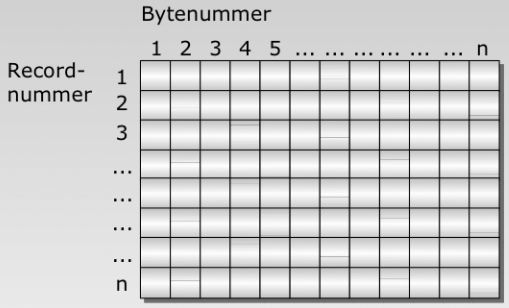
Hardware Token, Markus Winkler 29

Datenstrukturen von EF



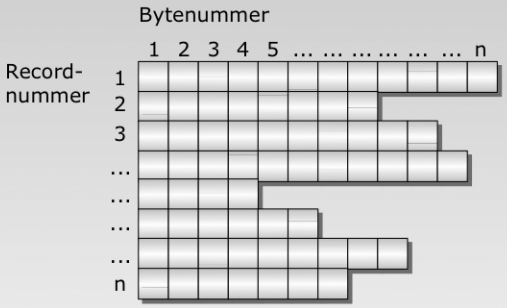
Hardware Token, Markus Winkler 30

Dateistruktur linear fixed



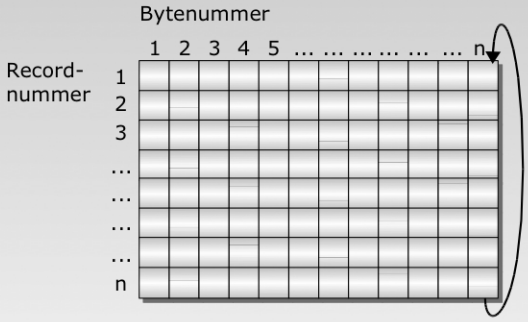
Hardware Token, Markus Winkler 31

Dateistruktur linear variable



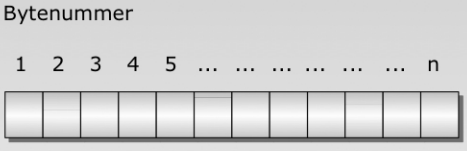
Hardware Token, Markus Winkler 32

Dateistruktur cyclic



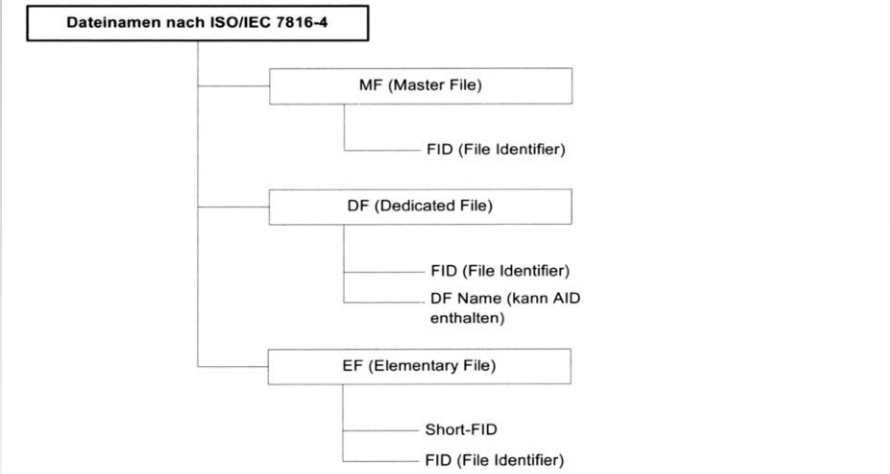
Hardware Token, Markus Winkler 33

Dateistruktur transparent



Hardware Token, Markus Winkler 34

Dateinamen



Klassifizierungsbaum der Dateinamen bei Chipkarten-Betriebssystemen nach ISO/IEC 7816-4.

* Abbildung nach Rankl W., Handbuch der Chipkarten, Hanser Verlag 2002

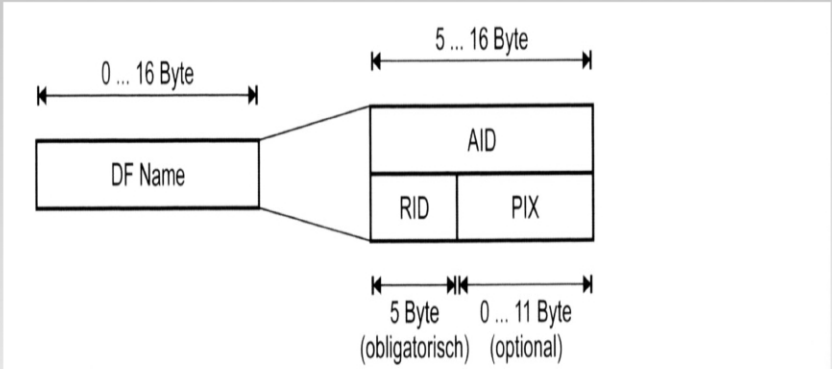
Hardware Token, Markus Winkler 35

Vorgegebene Dateinamen

FID	Name und Zweck	Norm
'2F00'	Diese FID ist reserviert für die Datei EF_{DIR} (*directory*) und wird zur Speicherung von Application Identifiers (AIDs) mit dazugehöriger Pfadangabe zur korrespondierenden Anwendung benutzt.	ISO/IEC 7816-4
'2F01'	Diese FID ist reserviert für die Datei EF_{ATR} mit den Erweiterungen zum ATR.	ISO/IEC 7816-4
'3F00'	Das MF ist das Wurzelverzeichnis für alle Dateien einer Chipkarte.	ISO/IEC 7816-4, GSM 11.11, TS 102.221, EMV
'3FFF'	Diese FID ist reserviert für die Dateiselektion durch Pfadangabe.	ISO/IEC 7816-4
'FFFF'	Diese FID ist reserviert für zukünftige Benutzung durch ISO/IEC.	ISO/IEC 7816-4

Hardware Token, Markus Winkler 36

Namen für DF



Der DF Name im Zusammenhang mit dem Aufbau des AID (application identifier) aus RID (registered identifier) und PIX (proprietary application identifier extension).

* Abbildung nach Rankl W., Handbuch der Chipkarten, Hanser Verlag 2002

Hardware Token, Markus Winkler 37

Aufbau der Dateinamen AID nach ISO 7816-5

RID			Bedeutung
D1	D2 ... D4	D5 ... D10	
X	...	...	Kategorie der Registrierung 'A' – Internationale Registrierung 'D' – Nationale Registrierung
...	X	...	Ländercode, Codierung nach ISO 3166
...	...	X	Nummer des Anwendungsherausgebers, wird von der nationalen bzw. internationalen Registrierungsinstanz vergeben

* Abbildung nach Rankl W., Handbuch der Chipkarten, Hanser Verlag 2002

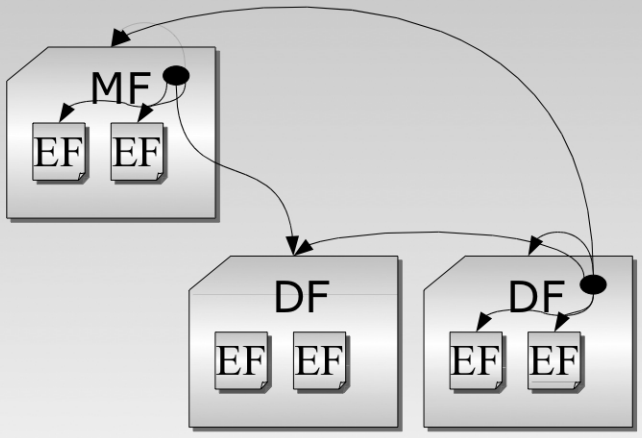
Hardware Token, Markus Winkler 38

EF_{DIR}	**Verzeichnis-EF (*dir – directory*)**	
Beschreibung	Diese Datei enthalt Informationen über die auf einer Chipkarte befindlichen Anwendungen.	
Datei	FID = '2F00'; Struktur: linear fixed, Dateigröße: n Bytes Zugriffe: READ: immer; UPDATE: je nach Anwendung, im Allgemeinen jedoch nur Administrator	
Codierung eines Records	Byte 1:	Template für Anwendung '61'
	Byte 2:	Länge des Template für Anwendung (3 . . . 127)
	Byte 3:	Kennzeichen des AID '4F'
	Byte 4:	Länge des AID (1 . . . 16)
	Byte 5 . . . n:	AID
	Byte n + 1:	Kennzeichen der Anwendungsbezeichnung (*application label*) '50'
	Byte n + 2:	Länge der Anwendungsbezeichnung
	Byte n + 3 . . . m:	Anwendungsbezeichnung in ASCII (0 . . . 16)
Beispiel	'61 0F 4F 05 D2 76 00 00 60 50 05 52 61 6E 6B 6C'	
	'61'	⇒ Template für Anwendung
	'0F'	⇒ Länge des Templates ⇒ Länge = 15 Byte
	'4F'	⇒ Kennzeichen des AID
	'05'	⇒ Länge des AID ⇒ Länge = 5 Byte
	'D2 76 00 00 60'	⇒ AID
	'4F'	⇒ Kennzeichen der Anwendungsbezeichnung
	'05'	⇒ Länge der Anwendungsbezeichnung
		⇒ Länge = 5 Byte
	'52 61 6E 6B 6C'	⇒ Anwendungsbezeichnung = „Rankl“

* Abbildung nach Rankl W., Handbuch der Chipkarten, Hanser Verlag 2002

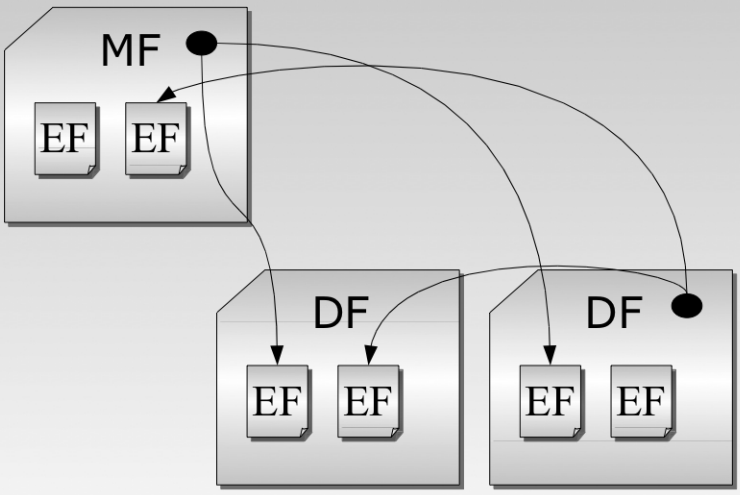
Hardware Token, Markus Winkler 39

Erlaubte Datei/Verzeichniszugriffe



Hardware Token, Markus Winkler 40

Unerlaubte Datei/Verzeichniszugriffe



Hardware Token, Markus Winkler 41

Inhalt der Dateideskriptoren

- Name der Datei: z.B. FID=0001
- Dateityp: z.B. EF
- Dateistruktur: z.B. linear fixed
- Dateigröße: z.B. 3 Records à 5 Byte
- Zugriffsbedingungen: Steuerung laut Zustand aus Zustandsautomat mit „<„, „≤“, „≥“, „>“, „=„ Bedingung, z.B. nur nach PIN-Eingabe
- Verbindung zum Dateibaum: z.B. direkt unter MF

Hardware Token, Markus Winkler 42

Dateiattribute

b8	b7	b6	b5	b4	b3	b2	b1	**Bedeutung**
0	0							Not sharable
0	1							Sharable
0		0	0	0				Working EF
0		0	0	1				Internal EF
0		1	1	1	0	0	0	DF
0					0	0	1	Transparent EF
0					0	1		Linear fixed EF
0					1	0		Linear variable EF
0					1	1		Cyclic EF
0							1	SIMPLE-TLV

* Abbildung nach Rankl W., Handbuch der Chipkarten, Hanser Verlag 2002

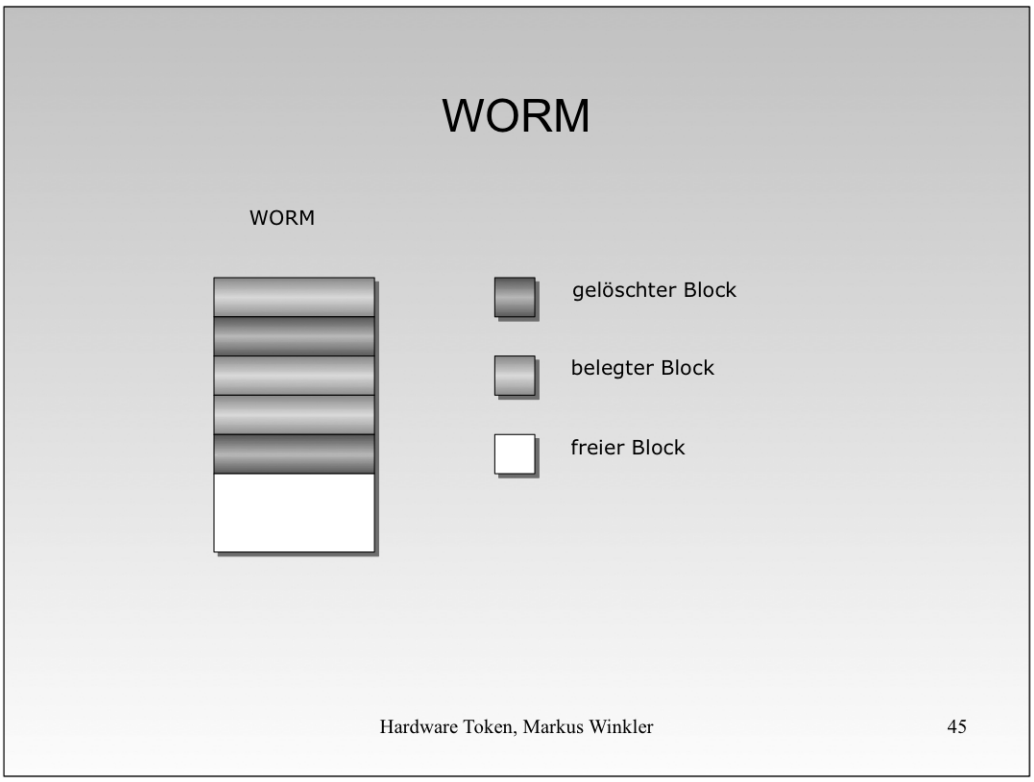
Hardware Token, Markus Winkler 43

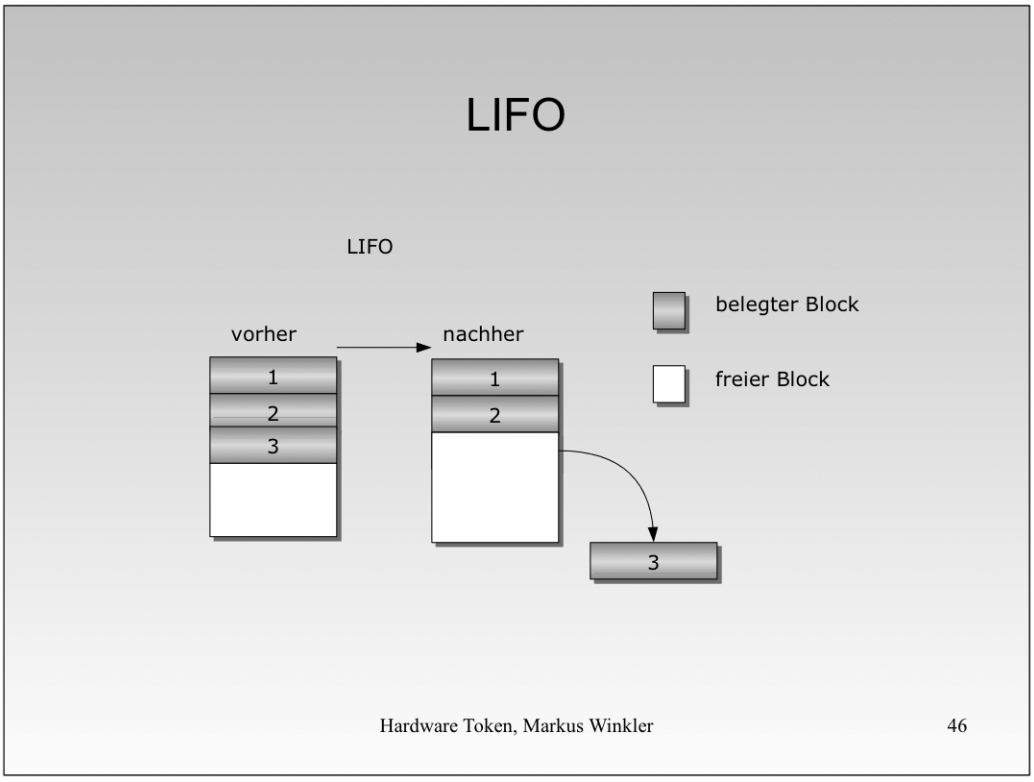
Inhalt der Sicherheitsattribute

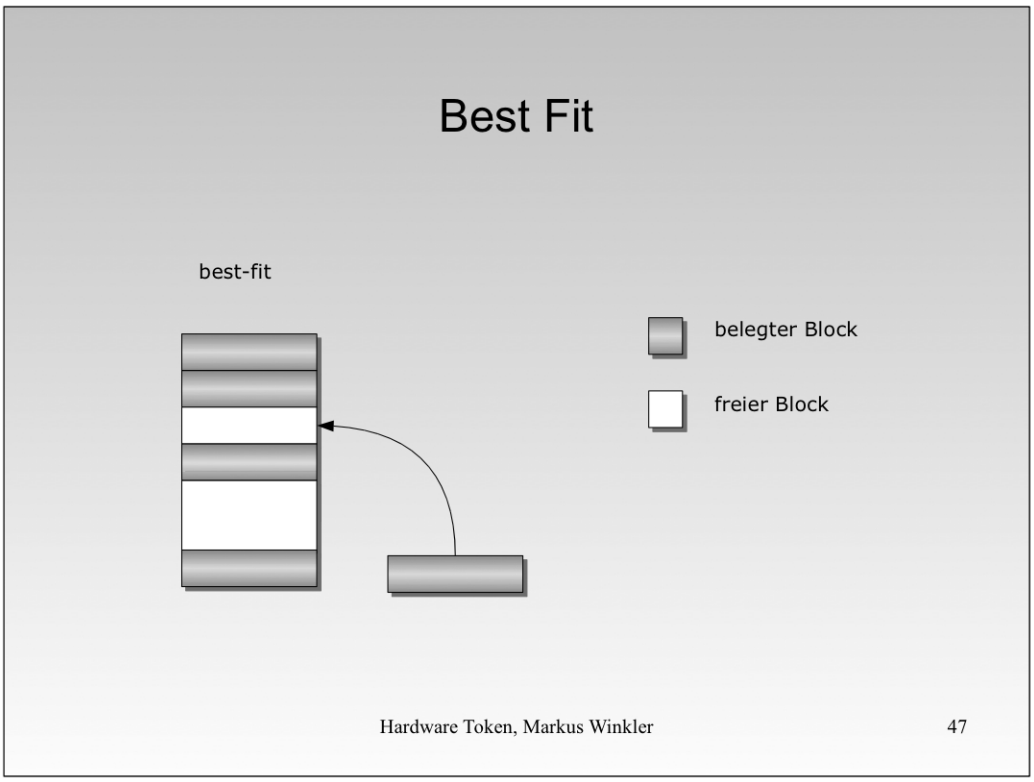
Beispiele von Sicherheitsattributen:

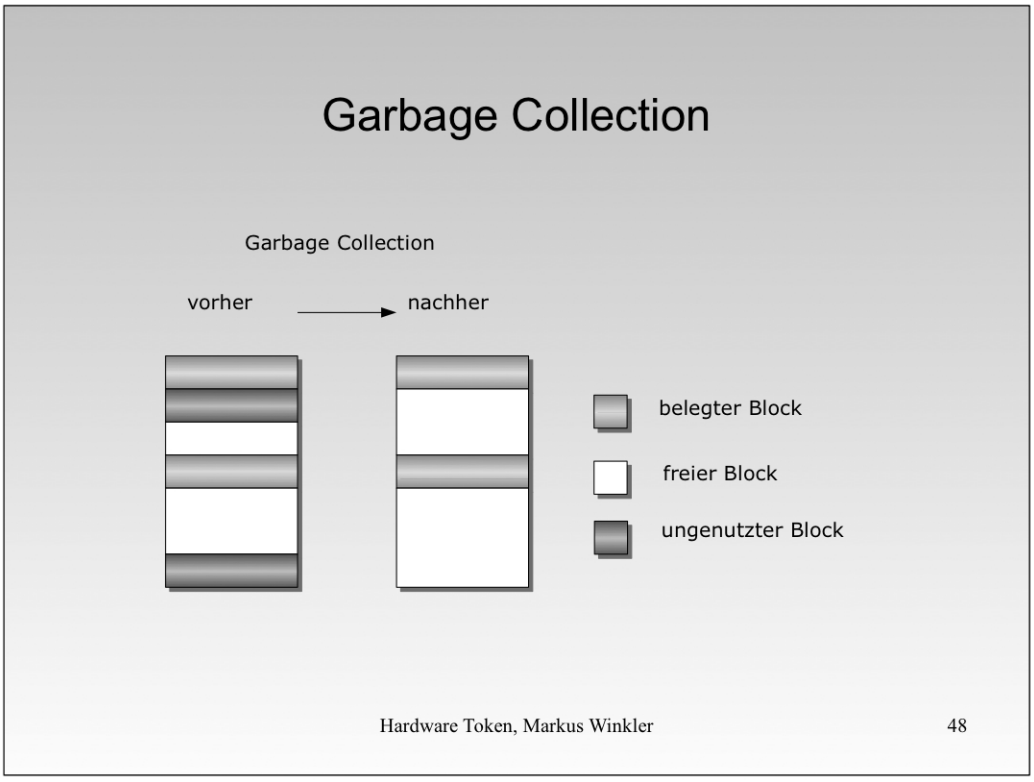
- WORM: nur einmaliges Schreiben möglich
- Atomar: immer atomare Verarbeitung der Daten
- Häufiges Schreiben: EEPROM-Schutz (Daten werden ständig in verschiedene Bereiche geschrieben, vor allem bei Terminalkarten wichtig)
- EDC-Code (Error detection code): Erkennung und Korrekturmöglichkeit von Bytefehlern im Speicher
- Gleichzeitiger Zugriff aus verschiedenen Kanälen
- Auswahl: kontaktlos und/oder kontaktbehaftet

Hardware Token, Markus Winkler 44

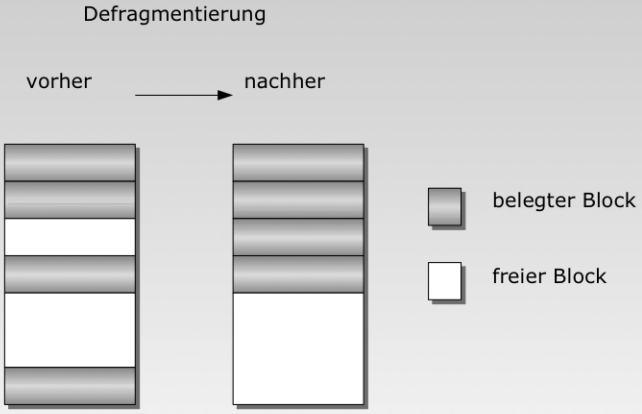

WORM
WORM
gelöschter Block
belegter Block
freier Block
Hardware Token, Markus Winkler
45


LIFO
LIFO
belegter Block
freier Block
vorher
nachher
1
2
3
1
2
3
Hardware Token, Markus Winkler
46


Best Fit
best-fit
belegter Block
freier Block
Hardware Token, Markus Winkler
47

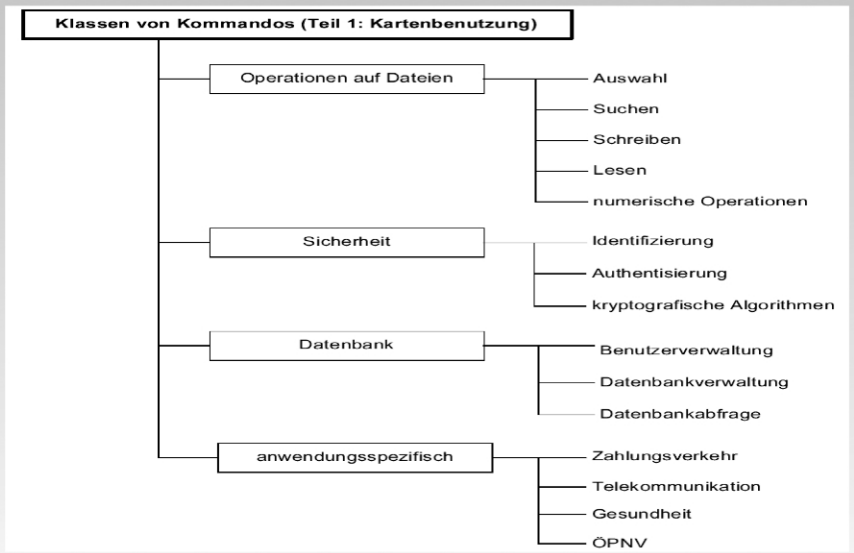

Garbage Collection
Garbage Collection
vorher
nachher
belegter Block
freier Block
ungenutzter Block
Hardware Token, Markus Winkler
48

Defragmentierung



Hardware Token, Markus Winkler 49

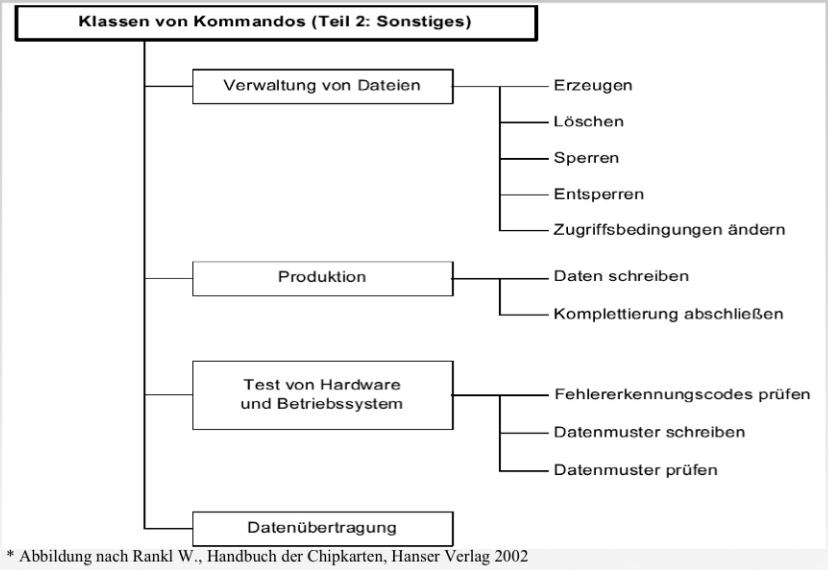
Chipkarten-Kommandos: Anwendungen



* Abbildung nach Rankl W., Handbuch der Chipkarten, Hanser Verlag 2002

Hardware Token, Markus Winkler 50

Chipkarten-Kommandos: Sonstige



* Abbildung nach Rankl W., Handbuch der Chipkarten, Hanser Verlag 2002

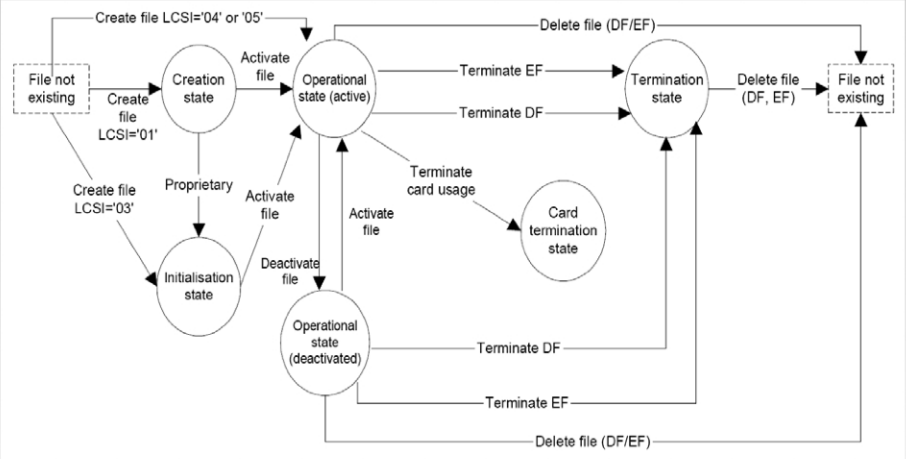
Hardware Token, Markus Winkler 51

Wichtige Kommandos auf Dateisystem

- Definiert nach ISO 7816-4 und ISO 7816-9
- Select File
- Transparentes Lesen und Schreiben:
  - Read Binary Update Binary
- Satz-basiertes Lesen und Schreiben
  - Read Record Update Record Append Record
- Create File und Delete File
- Deactivate und Activate File
- Increase und Decrease: Werterhöhung/Reduktion
- Get Data und Put Data für TLV-Codierungen

Hardware Token, Markus Winkler 52

Lebenszyklus von Dateien



* Abbildung nach Rankl W., Handbuch der Chipkarten, Hanser Verlag 2002

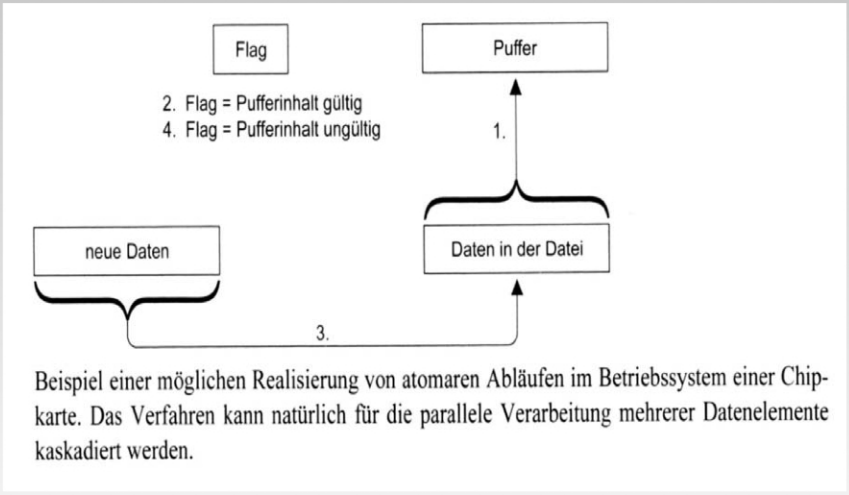
Atomare Abläufe von Kommandos

Da Chipkarten oftmals zu früh aus dem Terminal gezogen werden bzw. kontaktlose Chipkarten aus dem „Übertragungsfeld“ entfernt werden, sind überraschende Stromausfälle und Unterbrechungen der Kommunikation üblich

➔ Alle Abläufe in der Chipkarte müssen daher gegen diese ungeplanten Unterbrechungen vorbereitet sein.

Eine in Chipkarten dafür implementierte Lösung sind atomare Abläufe von Kommandos. Bei Dateien Attribut für atomare Abläufe vorhanden

Atomare Abläufe von Kommandos



Beispiel einer möglichen Realisierung von atomaren Abläufen im Betriebssystem einer Chipkarte. Das Verfahren kann natürlich für die parallele Verarbeitung mehrerer Datenelemente kaskadiert werden.

* Abbildung nach Rankl W., Handbuch der Chipkarten, Hanser Verlag 2002

Hardware Token, Markus Winkler 55

Zustandsautomat

- Bei Chipkarten kann die Reihenfolge von Kommandos vorgegeben werden kann. Es besteht also die Möglichkeit, alle Kommandos in ihren Parametern und ihrer Reihenfolge genau festzulegen
- Dies ist auch ein zusätzlicher Zugriffsschutz parallel zu den allgemeinen Zugriffsrechten auf Dateien
- Mit Zustandsautomaten kann man alle Kommandos mit allen Parametern vor der Ausführung innerhalb eines definierten Zustandsgraphen überprüfen
- Zustandsautomaten haben in der Chipkarte einige große Vorteile. Weil nur sehr wenige Kommandos in einer fest definierten Sequenz vorgegeben werden, benötigen sie wenig Speicher und Programmaufwand

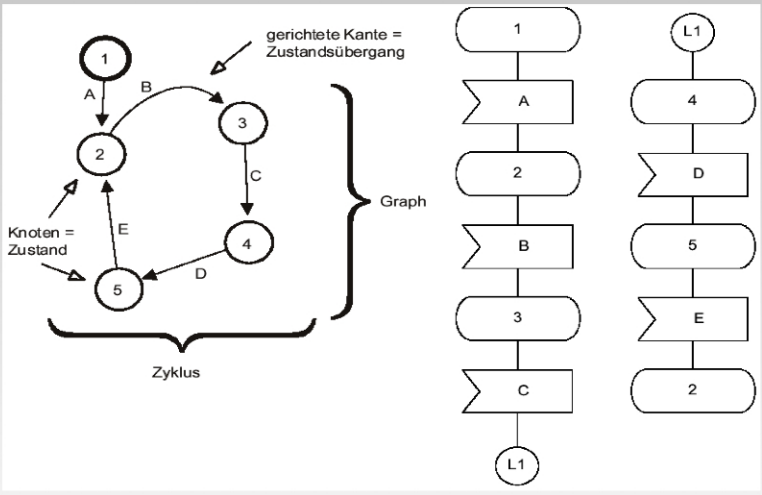
Hardware Token, Markus Winkler 56

Zustandsautomat

- Für viele Anwendungsfälle reicht es aus, die Dateiinhalte mit den objektorientierten Zugriffsmechanismen zu schützen und sonst alle Kommandos frei in ihrer Reihenfolge zuzulassen. Lediglich einige Abläufe, wie beispielsweise die Authentisierung müssen in ihrer Reihenfolge vorgeschrieben werden. Dies kann sehr speicherökonomisch durch einen einfachen Zustandsautomaten geschehen
- Die korrekte Beschreibung aller Abläufe und aller Kommandos zu einer Chipkarte ist aufwendig und muss oft teilweise empirisch ermittelt werden

Hardware Token, Markus Winkler 57

Zustandsautomat: Zustandsgraph



* Abbildung nach Rankl W., Handbuch der Chipkarten, Hanser Verlag 2002

Hardware Token, Markus Winkler 58

Zustandsautomat: Zustandsgraph in Tabellenform

	von Zustand				
nach Zustand	1	2	3	4	5
1	---	---	---	---	---
2	A	---	---	---	E
3	---	B	---	---	---
4	---	---	C	---	---
5	---	---	---	D	---

* Abbildung nach Rankl W., Handbuch der Chipkarten, Hanser Verlag 2002

Hardware Token, Markus Winkler 59

Angriffsmöglichkeiten auf Chipkarten

- Angriffe auf den Kartenkörper
- Angriffe auf die Hardware des Chips
- Angriffe auf das Betriebssystem
- Angriff auf die Anwendungen
- Angriffe durch Überlistung des Zugriffsschutzes: z.B. durch illegale Authentifizierung
- Angriffe durch Herausfinden der kryptografischen Schlüssel außerhalb der Chipkarte
- Angriff durch Chip-Austausch

Kartensysteme, SS 2008, E. Piller 60

Angriffszeitpunkte

- Während der Chipentwicklung
- Während der Chip-Produktion
- Während der Chip-Vorinitialisierung
- Während des Chiptransportes und der Chiplagerung
- Während der Chip-Initialisierung
- Während der Chip-Personalisierung
- Während der Auslieferung der Chipkarte an den Inhaber
- Beim Inhaber: im Stillstand oder bei der Benutzung
- Nach einem Diebstahl
- Nach einer temporären Sperre bzw. Entsperrung danach
- Nach der Beendigung der Lebensphase (dauerhafte Sperre, Gültigkeitsende, Zerstörung, ...)

Kartensysteme, SS 2008, E. Piller 61

B. Abkürzungsverzeichnis

ADF Application Dedicated Files

AID Application Identifier

AM Access Mode

ARR Access Rule References

ASCII American Standard Code for Information Interchange

ATM Automated Teller Machine

BIOS Basic Input Ouput System

DF Dedicated File

DIN Deutsche Industrie Norm

DIS Draft International Standard

EEPROM Electrically Erasable Programmable Read Only Memory

EF Elementary File

EPROM Erasable Programmable Read Only Memory

FAT File Allocation Table

FID File Identifier

GSM Global System for Mobile communication

IEC International Electrotechnical Commission

ISO International Organisation for Standardization

MF Master File

NTFS New Technology File System

OS Operating System

PC Personal Computer

POS Point Of Sale

PIN Personal Identification Number

PIX Properitary Application Identifier Extension

PTT Postes Télégraphes et Téléphones

RID Registered Identifier

UICC Universal Integrated Circuit Card

USB Universal Serial Bus

SC Security Condition

SFI Short File Identifier

SCQL Structured Card Query Language

SQL Structured Query Language

TS Technical Specification

TVL Type-Length-Value

WORM Write Once Read Multiple

C. Literaturverzeichnis

[ecard 2008] : *Österreichische e-card*. 2008. – URL http://www.chipkarte.at

[Bechtel 2006] BECHTEL, Uthelm: *Mac OS X 10.4 Tiger - Intel Edition*. Addison-Wesley, 2006

[Buchmann 2008] BUCHMANN, Johannes: *Einführung in die Kryptographie*. Springer, 2008

[Elmenreich 2002] ELMENREICH, Wilfried: *Systemnahes Programmieren, 3. Auflage*. Insitut für Technische Informatik, Technische Universität Wien, 2002

[ETSI GSM 11.11 1995] : *Specification of the Subscriber Identity Module*. 1995

[ETSI TS 102.221 2004] : *UICC-Terminal Interface; Physical and Logical Characteristics*. 2004

[ISO/IEC 7810 2003] : *Identification cards - Physical characteristics*. 2003

[ISO/IEC 7811-1 2002] : *Identification cards - Recording technique – Part 1: Embossing*. 2002

[ISO/IEC 7811-6 2002] : *Identification cards - Recording technique – Part 6: Magnet strip – High coercivity*. 2002

[ISO/IEC 7816-1 2003] : *Physical characteristics*. 2003

[ISO/IEC 7816-10 1999] : *Electronic signals and answer to reset for synchronous cards*. 1999

[ISO/IEC 7816-2 2004] : *Cards with contacts - Dimensions and location of contact*. 2004

[ISO/IEC 7816-3 2002] : *Cards with contacts - Electrical interface and transmission protocols*. 2002

[ISO/IEC 7816-4 2005] : *Organization, security and commands for interchange*. 2005

[ISO/IEC 7816-5 2004] : *Registration of application providers*. 2004

[ISO/IEC 7816-6 2004] : *Interindustry data elements for interchange*. 2004

[ISO/IEC 7816-7 1999] : *Interindustry commands for Structured Card Query Language (SCQL)*. 1999

[ISO/IEC 7816-8 2004] : *Commands for security operations*. 2004

[ISO/IEC 7816-9 2004] : *Commands for card management*. 2004

[Kemper 2004] KEMPER, A. Eickler / A.: *Datenbanksysteme, 5. Auflage*. Oldenbourg Wissenschaftsverlag, 2004

[Rankl 2007] RANKL, Wolfgang Effing / W.: *Handbuch der Chipkarten, 4. Auflage*. Hanser Fachbuchverlag, 2007

[Russinovich 2005] RUSSINOVICH, David A. Solomon / Mark E.: *Windows Internals, 4. Auflage*. Microsoft Press, 2005

[Stallings 2002] STALLINGS, William: *Betriebssysteme - Prinzipien und Umsetzung, 4. Auflage*. Pearson Studium, 2002

[Tanenbaum 1999] TANENBAUM, Andrew S.: *Computerarchitektur, 4. Auflage*. Prentice Hall, 1999

[Tischer 1994] TISCHER, Michael: *PC intern 4 - Systemprogrammierung*. Data Becker, 1994